Introducción a la Lógica y la Computación

Parte III: Lenguajes y Autómatas

Autor 1ra. Versión: Alejandro Tiraboschi

Autor 2da. Versión: Pedro Sánchez Terraf

Autores 3ra. Versión: Raul Fervari y Ezequiel Orbe

1 Modelos de Computación

A continuación estudiaremos un *modelo de computación* llamado *autómata*.

Definición 1.1 (Modelo de Computación)**.** Un *modelo de computación* es un modelo matemático que aproxima el funcionamiento de una computadora, y mediante el cual se pueden estudiar, desde un punto de vista teórico, las capacidades y limitaciones de la misma.

En particular, un modelo de computación nos permite responder con precisión preguntales trascendentales como:

- ¿Qué problemas puede resolver una computadora? (Computabilidad)
- ¿Cuáles son los problemas computacionalmente dificiles? (Complejidad)

Existen diferentes modelos de computación, entre los cuales podemos nombrar las Máquinas de Turing (Turing - 1936), los Autómatas Finitos (Rabin y Scott - 1950) y el Cálculo Lambda (Church - 1936).

Los primeros dos modelos (Máquinas de Turing y Autómatas Finitos) se suelen tipificar como *modelos de estados*, mientras que el Cálculo Lambda como *modelo funcional*.

Los modelos de computación difieren principalmente en su *poder computacional* (que funciones podemos computar con un modelo) y en qué nos permiten estudiar. Por ejemplo, las Máquinas de Turing (MT) y el Calculo Lambda se utilizan para estudiar *decidibilidad*, esto es, qué puede y qué no puede resolver una computadora y cuáles problemas son *decidibles* y cuáles son *indecidibles*. Un problema es decidible si existe un algoritmo que, con suficiente tiempo y memoria, siempre pueda resolverlo. Si no existe dicho algoritmo entonces decimos que el problema es indecidible.

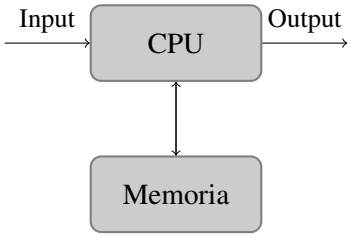


Figura 1: Modelo General de Máquina

1.1 Modelo General de Máquina

Para poner en contexto donde se encuentran los autómatas dentro del mundo de los modelos de computación, empezemos por definir de forma abstracta y simplificada lo que consideraremos como una *máquina* (computadora).

En su forma más básica, podemos pensar una computadora como compuesta por dos componentes: una memoria y una unidad central de procesamiento (CPU).

Nos interesa particularmente el tipo de memoria que tiene una máquina, ya que de eso dependerá su poder computacional. En función al tipo de memoria podemos distinguir los siguientes tipos de máquinas, ordenadas de menor a mayor poder computacional:

1. **Autómatas de Estados Finitos** (Finite State Automata): Son aquellas máquinas que *no* tienen de memoria temporaria.
2. **Autómatas con Pila** (Pushdown Automata): Son aquellas máquinas que tienen una *pila* como memoria temporaria.
3. **Máquinas de Turing** (Turing Machines): Son aquellas máquinas que tienen memoria de lectura y escritura de acceso aleatoria.

Cada tipo de máquina tiene distinto poder computacional, el cual se va incrementando a medida que el tipo de memoria es menos restrictivo. Como es de esperar, la complejidad también aumenta con el poder de computacional.

En esta materia no estudiaremos las Máquinas de Turing, pero sí veremos distintos tipos de Autómatas de Estados Finitos y con Pila, y estudiaremos sus propiedades.

2 Autómatas Finitos Determinísticos

2.1 Sistemas de estados finitos

En nuestra vida cotidiana nos encontramos con muchos sistemas que se pueden modelar como sistema de estados finitos. Los mismos siempre se encuentran en un estado determinado dentro de un conjunto *finito* de estados posibles, y la *transición* entre estados se da como consecuencia de recibir un determinado estímulo de entrada (un input). En estos sistemas, podemos considerar que la función de un estado es la de *recordar* una parte relevante de la historia del sistema.

Ejemplo 2.1. Consideremos un interruptor e intentemos modelarlo como un sistema de estados finitos. En su forma más simple podemos pensar un interruptor como un sistema con dos estados posibles,

un estado de **ON** y un estado de **OFF**, y en el cual la transición entre ambos estados ocurre cuando alguién *apreta* (push) el interruptor. La Figura 2 muestra un diagrama de este sistema.

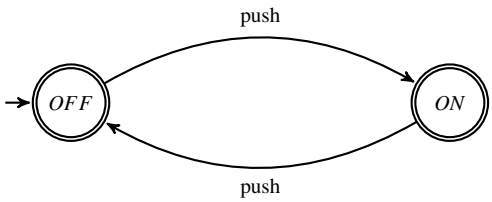


Figura 2: Modelo de Estados de un Interruptor

Cuando el interruptor está en el estado **ON**, podemos interpretarlo como "recordando" que alguien en algún momento lo apretó y lo puso en ese estado. Si luego de un cierto tiempo, lo observamos y el interruptor está en el estado **OFF**, entonces podemos interpretarlo como recordando que alguien en algún momento lo apretó y lo paso a ese estado.

Todos los sistemas de estados finitos comparten ciertas características comúnes. En particular, todos tienen:

1. Un *estado inicial*, en el cual el sistema se inicia,
2. un conjunto de *estados finales* (o de *aceptación*), que representan los estados a los cuales nos interesa llegar, y,
3. el efecto de aplicar una entrada (o input) depende únicamente del estado en el cual se encuentra el sistema cuando se la aplica.

Ejemplo 2.2. Consideremos un expendedor de golosinas, el cual entrega una golosina cuando se le ingresa cierta cantidad de dinero. Supongamos que el expendedor comienza su funcionamiento sin dinero, que solo entrega una golosina cuando le han ingresado $ 1 (un peso), y que acepta monedas de $ 0.25, $ 0.5 y $ 1. El expendedor de alguna forma debe recordar cuanto dinero le han ingresado para, así, saber cuando entregar la golosina.

Podemos modelar el expendedor de la siguiente forma:

- Con 5 estados, para cuando el expendedor tenga $ 0, $ 0.25, $ 0.5, $ 0.75 y $ 1 respectivamente.
- Inicialmente, el expendedor no tiene dinero, es decir, está en el estado $ 0.
- Con un solo estado final (el estado $ 1), que es cuando se entrega la golosina.
- Luego de entregar la golosina, se vuelve al estado inicial.
- Solo acepta monedas de $ 0.25, $ 0.5 y $ 1.

La Figura 3 presenta nuestro modelo del expendedor.

Ha de notarse que este modelo es una simplificación de un modelo real de un expendedor ya que no considera las situaciones donde se ingrese mas de $ 1, ¿cómo lo solucionarías?

Lo que acabamos de presentar son las características básicas de lo que se denominan *autómatas finitos determinísticos* y como habrán podido observar, hasta aquí hemos lidiado con conceptos y nociones sumamente intuitivas. A continuación, nuestro trabajo será definirlos y estudiarlos formalmente.

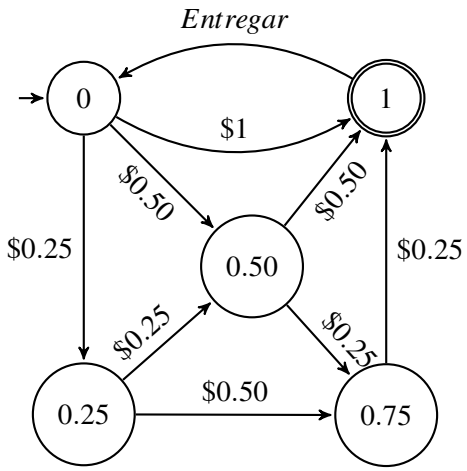


Figura 3: Modelo de un Expendedor de Golosinas

2.2 Autómata Finito Determinístico

Un autómata finito determinístico (DFA) es un modelo matemático de un sistema, con entradas y salidas discretas. Es *determinístico* ya que el sistema solo puede estar en una, entre un número finito, de configuraciones internas o *estados*.Además, el *estado* del sistema es la única información que se toma en cuenta para pasar a otro estado cuando el sistema recibe una entrada.

En ciencias de la computación es posible encontrar numerosos ejemplos de DFAs, y la teoría de autómatas es una herramienta sumamente útil para estudiar dichos sistemas. Programas tales como editores de texto y analizadores léxicos, encontrados en la mayoría de los compiladores, son a menudo modelados como sistemas de estados finitos.

Definición 2.1 (Autómata Finito Determinístico)**.** Un *autómata finito determinístico (DFA)* $M = (Q, \Sigma, \delta, q_0, F)$ es una 5-upla donde:

1. $Q = \{q_1, q_2, \ldots, q_m\}$ es un conjunto finito de *estados*.
2. $\Sigma = \{a_1, a_2, \ldots, a_n\}$ es un conjunto finito de *símbolos de entrada (o input)* .
3. $\delta : Q \times \Sigma \mapsto Q$ es la función (total) de transición de estados.
4. $q_0 \in Q$ es el *estado inicial*.
5. $F \subseteq Q$ son los *estados finales*.

Ejemplo 2.3. Consideremos el sistema de estados de un interruptor que vimos en el Ejemplo2.1 (Figura 2). Definimos el autómata $M = (Q, \Sigma, \delta, q_0, F)$, de la siguiente forma:

- $Q = \{ON, OFF\}$.
- $\Sigma = \{push\}$.
- $q_0 = OFF$.
- $F = \{ON, OFF\}$.
- δ definida como:

$$\begin{aligned} \delta(ON, push) &= OFF. \\ \delta(OFF, push) &= ON. \end{aligned}$$

Es común representar la función de transición mediante una tabla a doble entrada en la cual, buscando el cruce entre la entrada de un estado q y la entrada de un input a, sabemos como cambia el estado q cuando recibe el input a. En esta tabla es común marcar el estado inicial con el símbolo $\rightarrow$ y los estados finales con el símbolo $*$.

Ejemplo 2.4. Sea M un autómata finito determinístico con estados $\{q_0, q_1, q_2\}$, símbolos de entrada $\Sigma = \{a, b\}$ y las siguientes reglas de transición: el estado q_0 se transforma en q_1 si el input es a y en q_0 si el input es b; el estado q_1 se transforma en q_1 si el input es a y en q_2 si el input es b; finalmente, el estado q_2 se transforma en q_2 si el input es a y en q_0 si el input es b. Es decir δ, la función de transición, está definida por:

$$\begin{array}{llll} \delta(q_0,a) & = q_1, & \delta(q_0,b) & = q_0 \\ \delta(q_1,a) & = q_1, & \delta(q_1,b) & = q_2 \\ \delta(q_2,a) & = q_2, & \delta(q_2,b) & = q_0 \end{array}$$

Esta función puede ser representada de forma más compacta con la siguiente tabla:

	a	b
$\rightarrow * q_0$	q_1	q_0
$* \ q_1$	q_1	q_2
q_2	q_2	q_0

Ejemplo 2.5. Consideremos el modelo de estados del expendedor de golosinas del Ejemplo 2.2 (Figura 3). Definimos el autómata $M = (Q, \Sigma, \delta, q_0, F)$, de la siguiente forma:

- $Q = \{0, 0.25, 0.50, 0.75, 1\}$.
- $\Sigma = \{0.25, 0.50, 1, E\}$.
- $q_0 = 0$.
- $F = \{1\}$.
- δ definida como:

	0.25	0.50	1	E
$\rightarrow 0$	0.25	0.50	1	–
0.25	0.50	0.75	–	–
0.50	0.75	1	–	–
0.75	1	–	–	–
$* 1$	–	–	–	0

Se debe notar que el sistema del Ejemplo 2.5 no es un autómata en el sentido de la Definición 2.1 ya que hay estados que no aceptan ciertos inputs y la definición de un DFA requiere que la función de transición δ sea una función total. En la sección siguiente extenderemos el concepto de autómata, con lo cual este tipo de situaciones será admisible.Mientras tanto, ¿cómo modificarias el sistema para que cumpla con la definición de un DFA?

2.2.1 Diagramas de Transición

Como han podido observar en los ejemplos anteriores existe una estrecha relación entre un DFA y su representación gráfica. De hecho a todo DFA se le puede asignar un grafo, al cual llamaremos *diagrama de transición*.

Definición 2.2 (Algoritmo de Generación de Diagramas de Transición)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA, su diagrama de transición se construye de la siguiente forma:

1. Se agrega un nodo con etiqueta q_i, por cada estado $q_i \in Q$.

2. Marcamos al nodo correspondiente al estado inicial q_0 con una flecha que ingresa.

3. Marcamos los nodos correspondientes a los estados finales $q_i \in F$ con doble círculos.

4. Agregamos un eje etiquetado a del nodo q_i al nodo q_j si existe una transición $\delta(q_i, a) = q_j$.

Si tenemos un diagrama de transición hay un DFA asociado a el mismo; y por lo tanto de ahora en más no distinguiremos entre DFAs y sus diagramas de transición.

Ejemplo 2.6. Sea M un autómata con estados $\{q_0, q_1, q_2, q_3\}$, símbolos de input $\{0, 1\}$, estado inicial q_0 y estado final q_0, y donde la función de transición está dada por la Tabla 1.

	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

Tabla 1:

La Figura 4 presenta el diagrama de transición de M, generado usando el algoritmo de la Definición 2.2

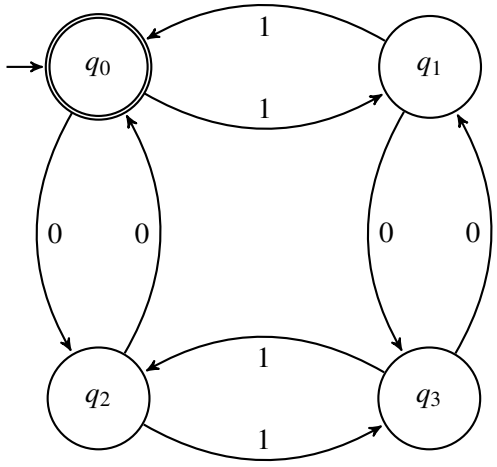


Figura 4: Diagrama de Transición

2.3 Autómatas y su Lenguaje

En esta sección estudiaremos la relación que existe entre los autómatas y los lenguajes. En particular nos interesan aquellas sucesiones de símbolos de entrada que nos llevan del estado inicial a un estado final, ya que ellas serán las que definan el *lenguaje del autómata*.

Comencemos por introducir las definiciones necesarias que nos permitirán formalizar el lenguaje de un autómata.

En lo que sigue, consideraremos a un *símbolo*, como una entidad abstracta que no definiremos formalmente, así como en geometría no se definen los *puntos* y las *líneas*. Letras y dígitos son ejemplos de símbolos frecuentemente usados.

Definición 2.3 (Alfabeto Σ, Cadena sobre Σ y Σ^*)**.** Un *alfabeto* (denotado por Σ) es un conjunto finito de símbolos. Una *cadena (o palabra) sobre el alfabeto* Σ es una secuencia finita de símbolos de Σ. La cadena vacía ε es la cadena con cero ocurrencias de símbolos. Σ^* denota el conjunto de todas las palabras sobre el alfabeto Σ.

Ejemplo 2.7. Sea el alfabeto $\Sigma = \{0,1\}$, luego 01, 000111 y 1010 son cadenas sobre el alfabeto Σ y, $\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots\}$.

Definición 2.4 (Longitud de una cadena)**.** La *longitud de una cadena w* (denotada por $|w|$) es el número de símbolos que la componen. La cadena vacía ε tiene longitud cero ($|\varepsilon| = 0$).

Definición 2.5 (Subcadena, Prefijo y Sufijo)**.** Una *subcadena* es una secuencia de símbolos incluida en una cadena.

Un *prefijo* de una cadena *w* es una subcadena de *w* que comienza con el primer símbolo de *w*. .

Un *sufijo* de una cadena *w* es una subcadena de *w* que termina con el último símbolo de *w*.

Definición 2.6 (Concatenación)**.** Dadas dos cadenas α y γ, su *concatenación* (denotada por αγ) es la cadena formada escribiendo la primera cadena seguida de la segunda sin espacios en el medio. Para toda cadena α vale que $\alpha\varepsilon = \varepsilon\alpha = \alpha$.

Ejemplo 2.8. Considere la cadena *bbabaa*. Luego, $|bbabaa| = 6$; la cadena *ab* es subcadena de *bbabaa*, mientras que la *aab* no lo es; la cadena *bba* es un prefijo de *bbabaa*, mientras que *ba* no lo es; la cadena *baa* es un sufijo de *bbabaa*, mientras que *aba* no lo es.

Dadas dos cadenas$w = aaa$ y $x = bbb$, luego, $wx = aaabbb$.

Finalmente podemos dar la definición de un lenguaje sobre un alfabeto.

Definición 2.7 (Lenguaje sobre un alfabeto Σ)**.** Un *lenguaje L sobre un alfabeto* Σ es un conjunto de palabras sobre el alfabeto Σ ($L \subseteq \Sigma^*$).

Ejemplo 2.9. El conjunto de *palindromes* o *capicúas* sobre el alfabeto $\{0,1\}$ es un lenguaje infinito. Algunos elementos de este lenguaje son ε, 0, 1, 00, 11, 000, 010, 101, 111.

Observación 2.1. Un lenguaje *L* sobre un alfabeto Σ no necesita incluir palabras con todos los símbolos de Σ, por lo tanto, una vez que establecimos que *L* es un lenguaje sobre Σ, también sabemos que será un lenguaje sobre toda extensión de Σ.

Para entender cual es la relación entre lenguajes y autómatas consideremos el autómata de la Figura 4:

Este autómata tiene como símbolos de entrada a $\Sigma = \{0,1\}$. Ahora consideremos las secuencias de entradas que nos llevan del estado inicial q_0 a cualquier otro estado. Por ejemplo, para llegar de q_0 a q_3 podemos hacerlo con las secuencias de símbolos de entradas $\langle 0,1\rangle$, ó $\langle 1,0\rangle$, ó $\langle 0,1,1,0,0,1\rangle$. Un análisis similar se puede realizar para los otros estados.

Si consideramos a Σ como un alfabeto, podemos interpretar a las secuencias de símbolos de entrada como palabras sobre este alfabeto. Por ejemplo, las secuencias $\langle 0,1\rangle$, $\langle 1,0\rangle$ y $\langle 0,1,1,0,0,1\rangle$ se pueden interpretar como las palabras 01, 10 y 011001 respectivamente. Luego podríamos preguntar a que estado nos lleva cada palabra sobre Σ. Por ejemplo, la palabra 11100 nos lleva de q_0 a q_1, mientras que la palabra 1001 nos lleva de q_0 nuevamente a q_0.

Es natural entonces, preguntarse cuales son las palabras que nos llevan desde el estado inicial a algún estado final. Anteriormente habiamos dicho que los estados finales o de aceptación encierran implícitamente una noción de *éxito*, por lo cual, si una palabra nos lleva del estado inicial a un estado

final podríamos decir que esa palabra es buena o que es *aceptada* por el autómata. Por ejemplo, la palabra 1001 es una palabra aceptada por el autómata.

Finalmente, el conjunto de palabras aceptadas por un autómata M es lo que se denomina como el *lenguaje aceptado por el autómata M*.

Lo interesante de conocer el lenguaje aceptado por un autómata, es que este lenguaje lo caracteriza (al autómata), por lo cual podemos estudiar diversas propiedades del mismo estudiando su lenguaje.

En lo que sigue formalizaremos la noción de lenguaje de un autómata.

Definición 2.8 (Transforma en)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA. Una *cadena en M* es un elemento de Σ^*. Sean $p, q \in Q$ estados de M y $\alpha = a_0 a_1 \dots a_n$ una cadena en M. Luego, diremos que α *transforma q en p* (denotado por $q \xrightarrow{\alpha} p$) si partiendo del estado q y aplicando sucesivamente las entradas a_0, a_1, hasta a_n, llegamos al estado p.

Observación 2.2. En lo que sigue nos interesan las siguientes propiedades de la relación *transforma en*.

- $p \xrightarrow{\varepsilon} p$ (ε transforma un estado en si mismo).
- Si $\alpha = a_0, a_1, \dots, a_n$:
  - $p \xrightarrow{\alpha} q \equiv p \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} q_n \xrightarrow{a_n} q$.
- Si $\alpha = \beta\gamma$, donde β y γ son cadenas, entonces:
  - $p \xrightarrow{\alpha} q$ sii existe $r \in Q$ tal que $p \xrightarrow{\beta} r \xrightarrow{\gamma} q$.

Definición 2.9 (Cadena aceptada por un autómata)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA y α una cadena en M. Diremos que α *es aceptada por M*, si existe $p \in F$ tal que $q_0 \xrightarrow{\alpha} p$, esto es, si α transforma el estado inicial q_0 en un estado final p.

Definición 2.10 (Lenguaje aceptado por un autómata)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un DFA, el *lenguaje aceptado por M* se define como:

$$L(M) = \{\alpha \in \Sigma^* | \text{ existe } p \in F \text{ tal que } q_0 \xrightarrow{\alpha} p\}.$$

Esto es, $L(M)$ es el conjunto de todas las cadenas aceptadas por el autómata.

Ejemplo 2.10. Determinenmos el lenguaje del autómata de la Figura 5:

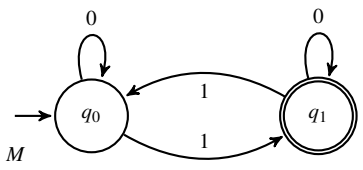


Figura 5: Autómata M_1

Luego de inspeccionar el autómata y de probar con algunas cadenas es fácil ver que el autómata solo acepta palabras con un número impar de 1's. Probemos formalmente que éste es el lenguaje de M_1.

Demostración. Sea w una cadena en M. Luego debemos probar la siguiente proposición:

$$q_0 \overset{w}{\to} q_1 \text{ si y solo si } w \text{ tiene un nro impar de } 1's.$$

Lo probamos por inducción en $|w|$.

Caso Base: $|w| = 0$ ($w = \varepsilon$)

($\to$): Trival, ya que $q_0 \overset{\varepsilon}{\to} q_0$ y por lo tanto $q_0 \overset{\varepsilon}{\to} q_1$ es falso.
($\leftarrow$): Trivial, ya que ε tiene un nro par de 1's (tiene cero 1's).

Paso Inductivo: Sea $w = xa$ con $|x| \geq 1$ y $a \in \Sigma$ y supongamos que vale la proposición para x.

($\to$): Si $q_0 \overset{xa}{\to} q_1$ entonces debemos considerar dos casos:

1. $a = 0$: Si $q_0 \overset{x0}{\to} q_1$, por definición de δ, $q_0 \overset{x}{\to} q_1 \overset{0}{\to} q_1$. Luego, por hipótesis inductiva (H.I.), x tiene un númeroro impar de 1's y por lo tanto $w = x0$ también tiene un número impar de 1's.

2. $a = 1$: Si $q_0 \overset{x1}{\to} q_1$, por definición de δ, $q_0 \overset{x}{\to} q_0 \overset{1}{\to} q_1$. Luego por H.I., x tiene un número par de 1's (por que de lo contrario tendríamos que $q_0 \overset{x}{\to} q_1$), y por consiguiente $w = x1$ tiene un número impar de 1's.

($\leftarrow$): Tenemos que analizar dos casos:

1. $a = 0$: Si $w = x0$ tiene un número impar de 1's, entonces x tiene un número impar de 1's y por H.I., $q_0 \overset{x}{\to} q_1$. Luego, por definición de δ, $q_1 \overset{0}{\to} q_1$ y se sigue que $q_0 \overset{x0}{\to} q_1$.

2. $a = 1$: Si $w = x1$ tiene un número impar de 1's, entonces x tiene un número par de 1's y por H.I., $q_0 \overset{x}{\to} q_0$. Luego, por definición de δ, $q_0 \overset{1}{\to} q_1$ y se sigue que $q_0 \overset{x1}{\to} q_1$.

□

Ejemplo 2.11. Probemos que el lenguaje del autómata del Ejemplo 2.6 es el conjunto de cadenas que tienen un número par de 0's y un número par de 1's.

Demostración. En este caso, para probar el lenguaje del autómata usaremos un *argumento geométrico*.

Si observamos el diagrama de transición correspondiente notamos que ir de izquierda a derecha o de derecha a izquierda significa aplicar el input 1. Ir de arriba a abajo o de abajo a arriba significa aplicar el input 0. Ahora bien, como q_0 es el estado inicial y final, una palabra aceptada va "recorriendo" el diagrama de tal forma que cuando termina subió tantas veces como bajó y fue a la izquierda tantas veces como fue a la derecha. Es decir que esta palabra tiene un número par de 0's y un número par de 1's.

Por otro lado, si una palabra tiene un número par de 0's y un número par de 1's es claro, por las mismas razones expuestas arriba, que termina su recorrido en q_0.

Observemos que ε es una cadena aceptada, debido a que q_0 es estado final y que $q_0 \overset{\varepsilon}{\to} q_0$. Claramente, la cadena ε tiene un número par de ceros y unos, pues tiene 0 ceros y 0 unos.

□

Como hemos mencionado, los autómatas estan caracterizados por su lenguaje, por lo cual si queremos comparar dos autómatas podríamos hacerlo comparando sus lenguajes. En otras palabras, la igualdad entre los lenguajes de dos autómatas nos sirve como noción de equivalencia entre los autómatas.

Definición 2.11 (Equivalencia de autómatas)**.** Sean M y M' dos DFA. Diremos que M y M' son equivalentes si $L(M) = L(M')$.

Ejemplo 2.12. Probemos que los autómatas M_1 y M_2 de la Figura 6 son equivalentes.

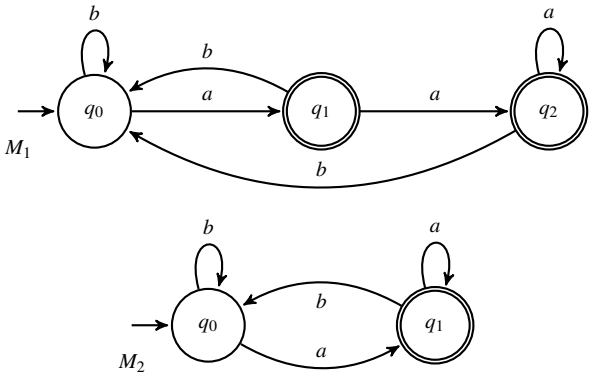


Figura 6: Autómatas M_1 y M_2

Inspeccionando los autómatas, es fácil determinar que el lenguaje aceptado por ambos autómatas es el de todas las palabras terminadas en a, esto es, $L(M_1) = L(M_2) = \{w \mid w \text{ termina en } a\}$, y por lo tanto que ambos autómatas son equivalentes.

Probemos formalmente que ambos autómatas son equivalentes.

Demostración. Para ambos autómatas debemos probar la siguiente proposición:

$$w \text{ es aceptada si y solo si } w \text{ termina en } a.$$

Prueba para M_1: Lo probamos por inducción en $|w|$.

Caso Base: $|w| = 0$ ($w = \varepsilon$)

($\rightarrow$): Trival, ya que $q_0 \xrightarrow{\varepsilon} q_0$ y por lo tanto ε no es aceptada.
($\leftarrow$): Trivial, ya que ε no termina en a.

Paso Inductivo: Sea $w = xz$ con $|x| \geq 1$ y $z \in \Sigma$ y supongamos que vale la propiedad para x.

($\rightarrow$): Si xz es aceptada entonces $q_0 \xrightarrow{xz} q_1$ (o $q_0 \xrightarrow{xz} q_2$). Por propiedad de $\rightarrow$, debe existir q_x tal que $q_0 \xrightarrow{x} q_x \xrightarrow{z} q_1$ (o $q_0 \xrightarrow{x} q_x \xrightarrow{z} q_2$). Luego, tenemos que considerar dos casos:

1. x es aceptada: Por H.I. x termina en a por lo cual debe ser que $q_0 \xrightarrow{x} q_1$ (o $q_0 \xrightarrow{x} q_2$). Luego, por definición de δ, $w = xz$ solo es aceptada si $z = a$ ya que $q_1 \xrightarrow{a} q_2$ (y $q_2 \xrightarrow{a} q_2$).

2. x no es aceptada: Por H.I. x no termina en a por lo cual debe ser que $q_0 \xrightarrow{x} q_0$. Luego, por definición de δ, $w = xz$ solo es aceptada si $z = a$ ya que $q_0 \xrightarrow{a} q_1$.

($\leftarrow$): Si $w = xa$, queremos ver que $q_0 \xrightarrow{xa} q_1$ (o $q_0 \xrightarrow{xa} q_2$). Por propiedad de $\rightarrow$, debe existir q_x tal que $q_0 \xrightarrow{x} q_x \xrightarrow{a} q_1$ (o $q_0 \xrightarrow{x} q_x \xrightarrow{a} q_2$). Luego, tenemos que considerar 2 casos:

1. x termina en a: Por H.I. x es aceptada, por lo cual debe ser que $q_0 \xrightarrow{x} q_1$ (o $q_0 \xrightarrow{x} q_2$). Luego por definición de δ tenemos que $q_1 \xrightarrow{a} q_2$ (y $q_2 \xrightarrow{a} q_2$), por lo cual xa es aceptada.

2. $\underline{x \text{ no termina en } a}$: Por H.I. x no es aceptada, por lo cual debe ser que $q_0 \xrightarrow{x} q_0$. Luego, por definición de δ, $q_0 \xrightarrow{a} q_1$ y por lo tanto $q_0 \xrightarrow{xa} q_1$, con lo cual $w = xa$ es aceptada.

Prueba para M_2:

$\underline{(\rightarrow)}$: Si $w = xz$ es aceptada entonces $q_0 \xrightarrow{xz} q_1$, si y solo si, por defincón de δ, $w = xa$ ya que la única forma de llegar a q_1 es a través de a.

$\underline{(\leftarrow)}$: Si $w = xa$ luego, por definición de δ, tenemos que $q_0 \xrightarrow{x} q_x \xrightarrow{a} q_1$, y por lo tanto $w = xa$ es aceptada.

Finalmente ahora probamos que $L(M_1) = L(M_2)$. Para ello debemos ver $L(M_1) \subseteq L(M_2)$ y que $L(M_1) \supseteq L(M_2)$.

$\underline{(\subseteq)}$: Si $\alpha \in L(M_1)$ entonces $\alpha = xa$ (termina en a), luego $\alpha \in L(M_2)$.

$\underline{(\supseteq)}$: Si $\alpha \in L(M_2)$ entonces $\alpha = xa$ (termina en a), luego $\alpha \in L(M_1)$.

□

3 Autómatas Finitos No Determinísticos

En la sección anterior definimos los autómatas finitos determinísticos (DFA) y vimos que todo DFA define un lenguaje. En particular, vimos que lo *determinístico* en un DFA se debe que, en todo momento, el estado del mismo esta completamente determinado. Esto se debe a que la función de transición es una función total que define exactamente, para todo estado y todo símbolo de entrada, a que estado se hace la transición. Ahora, ¿qué pasaría si permitimos 0,1 o más transiciones por símbolo?

Ejemplo 3.1. Consideremos el autómata de la Figura 7. ¿En que estado está el autómata si estando en q_0 aplicamos el símbolo de entrada 0? o, en otras palabras, ¿cuánto vale $\delta(q_0, a)$?.

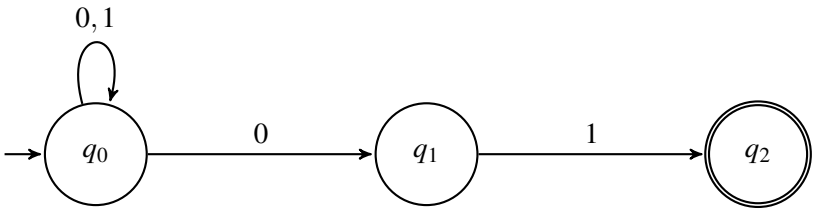


Figura 7: Autómata no determinístico

En este caso ya no podemos decir que el autómata esta en un estado determinado, ya que $\delta(q_0, a) = q_0$ y $\delta(q_0, a) = q_1$ al mismo tiempo.

El ejemplo anterior nos muestra que si ahora permitimos cero ó más transiciones por símbolo, perdemos la capacidad de establecer el estado exacto de un autómata, ya que el mismo ahora se encuentra, al mismo tiempo, en un *conjunto de estados*, perdiendo de esta manera su determinismo y transformándose así en un *autómata finito no determinístico*.

Definamos ahora formalmente a un autómata finito no determinístico.

Definición 3.1 (Autómata Finito No-Determinístico)**.** Un autómata finito no determinístico (NFA) $M = (Q, \Sigma, \delta, q_0, F)$ es una 5-tupla donde:

1. $Q = \{q_1, q_2, \ldots, q_m\}$ es un conjunto finito de *estados*.
2. $\Sigma = a_1, a_2, \ldots, a_n$ es un conjunto finito de *símbolos de entrada (o input)*.
3. $\delta : Q \times \Sigma \mapsto \mathcal{P}(Q)$ es la función de transición de estados.
4. $q_0 \in Q$ es el *estado final*.
5. $F \subseteq Q$ son los *estados finales*.

Es importante notar que la única diferencia entre la Definición 3.1 de un NFA y la Definición 2.1 de un DFA, es que, en un NFA la función de transición δ no es total y en vez de retornar un estado retorna un conjunto de estados.

Al igual que en el caso de un DFA, podemos representar la función de transición δ mediante una tabla de transición, donde para cada cruce entre un estado q y un símbolo de entrada a, tendremos el conjunto de estados al cual pasa el autómata.

Ejemplo 3.2. Consideremos el diagrama de transición de la Figura 7. Definimos el NFA $M = (Q, \Sigma, \delta, q_0, F)$ de la siguiente forma:

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0, 1\}$
- $F = \{q_2\}$
- q_0 es el estado inicial
- δ está definida por:

	0	1
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\emptyset$

La pregunta que surge ahora es: ¿cómo procesa un NFA una cadena de entrada?, es decir, ¿qué hace cuando tiene más de una transición posible para un mismo símbolo de entrada?

Ejemplo 3.3. Consideremos el NFA de la Figura 7 y veamos como procesa la cadena 00101.

El autómata inicia en q_0, su estado inicial. Cuando recibe el primer símbolo 0, hace una transición a los estados q_1 y q_0, es decir, el autómata está en los estados $\{q_0, q_1\}$. Esto se puede ilustrar en términos más concretos, si pensamos que lo que hace el autómata es crear *dos hilos de ejecución*, uno en el cual el autómata tiene está en el estado q_0, y otro en el cual el autómata está en el estado q_1. La Figura 8 representa esta situación.

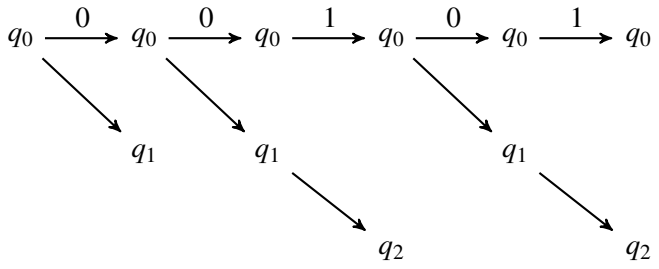


Figura 8: Como procesa un NFA una cadena de entrada

Continuemos procesando la cadena de entrada y veamos que sucede cuando el autómata recibe otro símbolo 0. En este caso, en el hilo de ejecución de q_0, nuevamente podemos hacer una transición

hacia q_1 y q_0, creando de esta forma otros dos hilos de ejecución. Sin embargo, el hilo de ejecución correspondiente a q_1 no tiene especificado como procesar el símbolo 0, y por lo tanto *aborta*. De esta forma, luego de procesar el segundo símbolo 0 estamos en el estado $\{q_0, q_1\}$.

Procesemos ahora el símbolo 1. En este caso en el hilo de ejecución de q_0 solo podemos hacer una transición hacia q_0, mientras que en el hilo de ejecución de q_1 se hace una transición hacia el estado q_2. Ahora el autómata está en el estado $\{q_0, q_2\}$.

Procesemos ahora el símbolo 0. En este caso, en el hilo de ejecución de q_0 hacemos transiciones a q_0 y q_1, mientras que en el hilo de ejecución de q_2 no se puede procesar ese símbolo, y por lo tanto aborta.

Finalmente procesamos el último símbolo 1. En este caso, el autómata termina en los estados $\{q_0, q_2\}$. Pregunta: ¿Ha sido aceptada la cadena 00101?

Definamos ahora el lenguaje aceptado por un NFA.

Definición 3.2 (Transforma en)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un NFA y $\alpha = a_0, a_1, \ldots, a_n \in \Sigma^*$. Sean $p, q \in Q$ estados de M. Diremos que α *transforma p en q* (denotado por $p \xrightarrow{\alpha} q$) si existen $q_1, \ldots, q_n \in Q$ tal que

$$q_1 \in \delta(p, a_0), q_2 \in \delta(q_1, a_1), \ldots, q_{i+1} \in \delta(q_i, a_i), \ldots, \\ q_n \in \delta(q_{n-1}, a_{n-1}), q \in \delta(q_n, a_n).$$

Observación 3.1. Es importante tener presente las siguientes propiedades de la relación *transforma en*:

- α puede transformar a un estado q_i en varios estados.
- α genera distintos recorridos desde q_0.

Como es de esperar, una cadena α será *aceptada* si alguno de los recorridos que genera desde q_0 termina en un estado final.

Se debe tener presente que la noción de aceptación, ahora tiene en cuenta que en un NFA trabajamos con conjuntos de estados, por lo cual una cadena α, pueden generar muchos recorridos desde el estado inicial. Algunos de estos recorridos no necesariamente terminan en un estado final e incluso pueden abortar tempranamente. Sin embargo, lo que nos interesa para decidir si una cadena es aceptada o no, es que dentro de todos esos posibles caminos, *al menos uno* termine en un estado final.

Definición 3.3 (Lenguaje aceptado por un autómata)**.** Sea $M = (Q, \Sigma, \delta, q_0, F)$ un NFA, el *lenguaje aceptado por M* se define como:

$$L(M) = \{\alpha \in \Sigma^* | \text{ existe } p \in F \text{ tal que } q_0 \xrightarrow{\alpha} p\}.$$

Ejemplo 3.4. Consideremos el NFA de la Figura 7. Es fácil ver que el autómata acepta solo las cadenas que terminan en 01. Es decir,

$$L(M) = \{w \mid w \text{ termina en } 01\}.$$

Demostración. Por inducción en $|w|$. □

3.1 Equivalencia entre DFAs y NFAs

Revisando la Definición 3.1 de un NFA, es fácil ver que la misma es una generalización de la Definición 2.1 de un DFA.

Esto nos lleva a preguntarnos si esta generalización permite a los NFAs reconocer más lenguajes que los DFAs.

Sin embargo, veremos a continuación que este no es el caso y que los NFAs reconocen los mismos lenguajes que los DFAs, esto es, veremos que para todo NFA existe un DFA que reconoce el mismo lenguaje, y viceversa, lo cual nos permitirá probar que ambas nociones son equivalentes.

El proceso de construir un DFA a partir de un NFA se denomina *determinización*. Comencemos por definir como construir un DFA a partir de un NFA utilizando una construcción llamada *construcción por subconjuntos*.

Definición 3.4 (Construcción por Subconjuntos)**.** Sea $N = (Q, \Sigma, \delta, q_0, F)$ un NFA. Construiremos el DFA, $D = (Q', \Sigma', \delta', q_0', F')$, de la siguiente forma:

1. $Q' = \mathcal{P}(Q)$.
2. $\Sigma' = \Sigma$.
3. $F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$.
4. $q_0' = \{q_0\}$.
5. $\delta'(q', a) = \{p \in Q \mid \text{ existe } q \in q' \text{ t.q. } q \xrightarrow{a} p\} = \bigcup_{q \in q'} \delta(q, a)$.

Observación 3.2. Es importante tener presente lo siguiente:

- $\delta'(q', a)$ el conjunto de estados accesibles, a través de a, desde cada $q \in q'$.

Ejemplo 3.5. Consideremos el autómata de la Figura 6. La Tabla 2 representa la tabla de transición de estados del DFA $D = (Q', \Sigma, \delta', q_0', F')$ obtenido mediante la construcción por subconjuntos. La Figura 9 presenta el diagrama de transición de estados del DFA D.

		0	1
(0)	$\emptyset$	$\emptyset$	$\emptyset$
(1) $\rightarrow$	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
(2)	$\{q_1\}$	$\emptyset$	$\{q_2\}$
(3) $*$	$\{q_2\}$	$\emptyset$	$\emptyset$
(4)	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
(5) $*$	$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
(6) $*$	$\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
(7) $*$	$\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Tabla 2: Tabla de transición de estados del autómata D.

A partir del Ejemplo 3.5 podemos ver que uno de los inconvenientes de la construcción por subconjuntos es la explosión en el número de estados que tiene el DFA resultante. De hecho, si $|Q| = n$

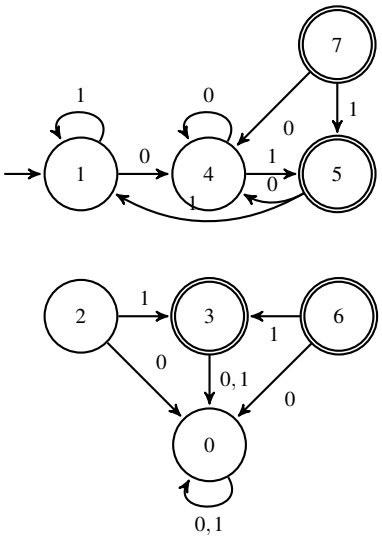


Figura 9: Diagrama de Transición de estados del DFA D.

entonces $|Q'| = 2^n$!!!. Sin embargo, el ejemplo también nos muestra que no todos los estados del DFA son alcanzables desde el estado inicial, y por lo tanto no son necesarios.

A continuación presentamos un algoritmo más eficiente que nos permitirá construir DFA que contenga solo los estados necesarios.

Definición 3.5 (Construcción Lazy)**.** Sea $N = (Q, \Sigma, \delta, q_0, F)$ un NFA. Construiremos el DFA, $D = (Q', \Sigma', \delta', q_0', F')$, de la siguiente forma:

1. $\Sigma' = \Sigma$.
2. $F' = \{q' \in Q' \mid q' \cap F \neq \emptyset\}$.
3. $q_0' = \{q_0\}$.
4. $\delta'(q', a) = \{p \in Q \mid \text{ existe } q \in q' \text{ t.q. } q \xrightarrow{a} p\} = \bigcup_{q \in q'} \delta(q, a)$.

Donde el conjunte de estados Q' se construye de la siguiente forma:

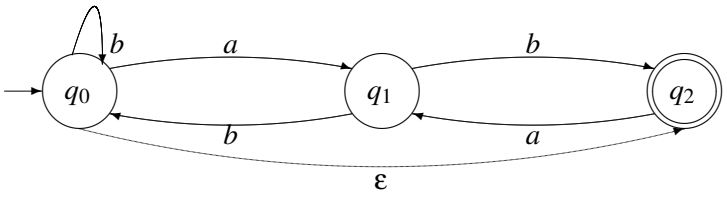
Data: estado inicial q_0, función δ, alfabeto Σ
Result: conjunto de estados Q'
$Q' \leftarrow \{\{q_0\}\}$;
$Q'' \leftarrow \emptyset$;
for $q' \in (Q' \backslash Q'')$ **do**
  for $a \in \Sigma$ **do**
    $Q' \leftarrow Q' \cup \{\bigcup_{q \in q'} \delta(q, a)\}$;
  end
  $Q'' \leftarrow Q'' \cup \{q'\}$;
end

3.2 Autómatas Finitos No Determinísticos con ε-transiciones

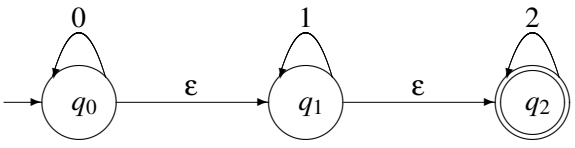
Ahora introduciremos la noción de autómata finito no determinístico. No es difícil ver que un conjunto aceptado por un autómata no determinístico también es aceptado por uno determinístico y viceversa. Sin embargo, el autómata finito no determinístico resultará útil para probar ciertos resultados y para ciertas aplicaciones. Daremos una definición no formal de este tipo de autómatas. El lector interesado en la definición formal puede consultar la subsección 3.2.1.

Modifiquemos la definición de autómata finito permitiendo cero, una, o más transiciones de un estado con el mismo símbolo de input. Es decir de un estado pueden partir un número arbitrario de flechas cada una etiquetada con cualquier símbolo de input. También permitamos transiciones sin inputs. Esta nueva definición nos da los *autómatas finitos no determinísticos (NFA) con* ε*-movimientos*. A nivel de diagramas de transición los grafos que representan estos autómatas serán de la siguiente forma: de cada nodo puede partir ninguna, una o varias flechas con la misma etiqueta de input. Además puede haber flechas que no se correspondan a ningún input, los ε-movimientos. A estas flechas les pondremos la etiqueta ε. Como en el caso de los autómatas determinísticos los NFA tienen un estado inicial y estados finales, que se denotarán en el diagrama de transición de la misma forma que en el caso determinístico

Ejemplo 3.6. El siguiente es el diagrama de un autómata finito no determinístico con ε-movimientos: los estados son q_0, q_1 y q_2. Los símbolos de input son a y b. El estado inicial es q_0 y el estado final es q_2.



Ejemplo 3.7. En este NFA los estados son q_0, q_1 y q_2. Los símbolos de input son 0, 1 y 2. El estado inicial es q_0 y el estado final es q_2.



El nombre "no determinístico" viene del hecho que si un estado recibe un input, entonces no está determinado cual es el próximo estado, si no que hay ciertos estados posibles. Los ε-movimientos reflejan la posibilidad de cambio de estado sin que necesariamente haya un input. Observar también, que existe la posibilidad que dado un estado no salga ninguna flecha de él. Esto denotará que en este estado el autómata "aborta". Es claro que no es fácil imaginarse una "máquina" que se comporte de esta manera, sin embargo el concepto de NFA con ε-movimientos resulta muy útil en la teoría de lenguajes.

Las tablas que describirán las reglas de transición de los NFA con ε-mov serán similares a las tablas que hemos hecho para los autómatas finitos determinísticos. Las diferencias son que cuando

aplicamos un input a un estado obtenemos un conjunto de estados (hacia donde van las flechas) o $\emptyset$ en el caso que no salga ninguna flecha. Además debemos agregar una entrada ε en la tabla para reflejar los posibles cambios de estado que producen los ε-movimientos.

Ejemplo 3.8. La tabla de transición correspondiente al NFA con ε-mov del ejemplo 3.6 es dada por la siguiente tabla:

	a	b	ε
q_0	q_1	q_0	q_2
q_1	$\emptyset$	q_0, q_2	$\emptyset$
q_2	q_1	$\emptyset$	$\emptyset$

Ejemplo 3.9. La tabla de transición correspondiente al NFA con ε-mov del ejemplo 3.7 es dada por la siguiente tabla:

	0	1	2	ε
q_0	q_0	$\emptyset$	$\emptyset$	q_1
q_1	$\emptyset$	q_1	$\emptyset$	q_2
q_2	$\emptyset$	$\emptyset$	q_2	$\emptyset$

Veamos ahora el lenguaje asociado a un NFA con ε-movimientos.

Definición 3.6. Sea M un NFA con ε-movimientos con símbolos de input Σ. Si $\alpha \in \Sigma^*$, diremos que α *transforma* p en q y lo denotaremos $p \xrightarrow{\alpha} q$, si existen $a_0, a_1, \ldots, a_n \in \Sigma$ tal que $\alpha = a_1 a_2 \ldots a_n$ y tal que existe un recorrido por flechas con etiquetas $a_0, a_1, \ldots, a_n$ que va del estado p al estado q. En el recorrido se pueden intercalar arbitrariamente flechas con etiqueta ε.

Es conveniente también definir que ε transforma todo estado en sí mismo, es decir $q \xrightarrow{\varepsilon} q$.

Ejemplo 3.10. En el autómata del ejemplo 3.7 tenemos, por ejemplo, que $q_0 \xrightarrow{01} q_1$, pues hay un camino que lleva q_0 a q_0 por 0, q_0 a q_1 por ε y q_1 a q_1 por 1.

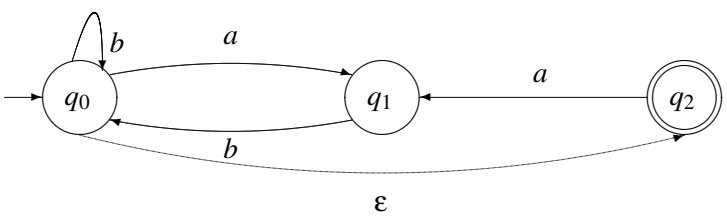
Observación 3.3. Si p y q estados, a símbolo de input y $p \xrightarrow{a} q$, no podemos asegurar que de p podemos pasar directamente a q por una flecha con etiqueta a. Por definición $p \xrightarrow{a} q$ significa que podemos llegar de p a q usando ε-movimientos además de una transición por a. Observando el autómata del ejemplo 3.7 vemos que $q_0 \xrightarrow{0} q_1$, pero no es cierto que hay una flecha con etiqueta 0 que manda q_0 a q_1.

En gran parte de las situaciones que necesitemos usar transiciones podremos tomarnos la licencia de usar la notación

$$p \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n \xrightarrow{a_n} q,$$

para indicar que hay un camino de flechas con etiquetas $a_0, \ldots, a_n$ que pasa por los estados $q_1, \ldots, q_n$ y que va de p a q (aquí permitiremos que los a_i puedan ser ε's).

Ejemplo 3.11. Sea M un NFA con ε-movimientos,con diagrama de transición :



Entonces, la tabla de transición es:

	a	b	ε
q_0	q_1	q_0	q_2
q_1	$\emptyset$	q_0	$\emptyset$
q_2	q_1	$\emptyset$	$\emptyset$

Veamos cuales son las transformaciones correspondientes a un input o un ε-movimiento:

$$\begin{array}{lll} q_0 \xrightarrow{\varepsilon} q_0, q_2 & q_0 \xrightarrow{a} q_1 & q_0 \xrightarrow{b} q_0, q_2 \\ q_1 \xrightarrow{\varepsilon} q_1 & q_1 \xrightarrow{b} q_0, q_2 & q_2 \xrightarrow{a} q_1 \\ q_2 \xrightarrow{\varepsilon} q_2 & q_2 \xrightarrow{a} q_1 & \end{array}$$

donde denotamos $p \xrightarrow{a} q, q'$ cuando $p \xrightarrow{a} q$ y $p \xrightarrow{a} q'$. La mayoría de las transformaciones son obvias y las otras se deducen de la definición. Sin embargo, creemos conveniente explicar algunas de ellas. Por ejemplo, recordemos que, por definición, $q \xrightarrow{\varepsilon} q$ para todo q estado, de allí que se justifica la primera columna de transformaciones. La transformación $q_1 \xrightarrow{b} q_2$ se deduce de que podemos llegar de q_1 a q_2 primero aplicando b y luego ε. En forma análoga se justifica $q_0 \xrightarrow{b} q_2$.

Como en el caso de autómatas determinísticos los estados finales nos determinan el lenguaje asociado al autómata.

Definición 3.7. Una cadena α es *aceptada* por el autómata si α transforma el estado inicial en uno final. Finalmente el *lenguaje aceptado por M* es el conjunto $L(M)$ de todas las cadenas aceptadas. Simbólicamente:

$$L(M) = \{\alpha \in \Sigma^* \mid q_0 \xrightarrow{\alpha} p, \text{ para algún } p \in F\}.$$

A nivel de diagramas de transición podemos entender que el lenguaje aceptado por M, un NFA con ε-movimientos, está formado por las cadenas $\alpha = b_0 b_1 \dots b_n$ tal que existe un recorrido por flechas con etiquetas $b_0, b_1, \dots, b_n$ que va del estado inicial a uno final, en el recorrido se pueden intercalar arbitrariamente flechas con etiqueta ε.

Observemos que como el ε es la cadena vacía el intercalar flechas con etiqueta ε no agrega nada a la palabra, y la palabra o cadena definitiva es aquella que no tiene ningún ε. También observemos que dada una cadena puede haber muchos recorridos partiendo del estado inicial, algunos de ellos pueden terminar en un estado final y otros no, sin embargo lo que interesa, para saber si una cadena es aceptada o no, es si por lo menos un recorrido alcanza un estado final.

Ejemplo 3.12. Averigüemos el lenguaje del autómata definido en el Ejemplo 3.7: observemos primero que si abandonamos el estado q_0 entonces vamos al estado q_1 y no podemos volver atrás. Análogamente si abandonamos el estado q_1, vamos al estado q_2 y ya no podemos salir de este estado. Luego si una cadena w alcanza el estado final, entonces comienza por q_0, pasa por q_1 y llega a q_2. Es decir que w es de la forma $0^i 1^j 2^k$ (i, j, k enteros mayores o iguales a 0). Recíprocamente una cadena de la forma $0^i 1^j 2^k$ puede alcanzar el estado final con el siguiente recorrido: $0, 0, \dots, 0$ (i-veces), ε, $1, 1, \dots, 1$ (j-veces), ε, $2, 2, \dots, 2$ (k-veces). Es decir que el lenguaje asociado a este autómata es $\{0^i 1^j 2^k : 0 \leq i, j, k\}$.

3.2.1 Formalización de los NFA

La definición formal de NFA es:

Definición 3.8. Un *autómata finito no determinístico con* ε*-movimientos (NFA con* ε*-mov)* es una 5-upla (Q,Σ,δ,q_0,F) que consiste de

1. un conjunto finito $Q=\{q_1,q_2,\ldots,q_n\}$ de *estados*,
2. un conjunto finito $\Sigma=\{a_1,a_2,\ldots,a_n\}$ de *símbolos de entrada* o *input*,
3. un conjunto de *reglas de transición* que transforma un estado con un input dado, en un conjunto posible de estados. También las reglas de transición describen la transformación de un estado en un conjunto de estados sin haber recibido input (ε-movimiento). Formalmente, las reglas de transición son dadas por una función $\delta : Q\times(\Sigma\cup\{\varepsilon\})\to\mathcal{P}(Q)$, donde $\mathcal{P}(Q)$ es el conjunto formado por los subconjuntos de Q.
4. Además, como en el caso del DFA, hay un *estado inicial* q_0 y un conjunto de estados finales F.

Veamos ahora el lenguaje asociado a un NFA con ε-movimientos:

Definición 3.9. Sea $M=(Q,\Sigma,\delta,q_0,F)$ un NFA con ε-movimientos. Si $\alpha\in\Sigma^*$, diremos que α *transforma* p en q y lo denotaremos $p\xrightarrow{\alpha}q$, si existen $a_0,a_1,\ldots,a_n\in\Sigma\cup\{\varepsilon\}$ y $q_1,\ldots,q_n$ en Q , tal que $\alpha=a_1a_2\ldots a_n$ y

$$q_1\in\delta(p,a_0),\ q_2\in\delta(q_1,a_1),\ \ldots q_{i+1}\in\delta(q_i,a_i),\ \ldots q_n\in\delta(q_{n-1},a_{n-1}),\ q\in\delta(q_n,a_n).$$

Es conveniente también definir que ε transforma todo estado en sí mismo, es decir $q\xrightarrow{\varepsilon}q$.

Observación 3.4. Sea $M=(Q,\Sigma,\delta,q_0,F)$ un NFA con ε-movimientos.

1. Como dijimos antes, si $a\in\Sigma$, entonces si $q\in\delta(p,a)$ tenemos que $p\xrightarrow{a}q$. Pero no necesariamente si $p\xrightarrow{a}q$, entonces se cumple que $q\in\delta(p,a)$. La definición de $p\xrightarrow{a}q$ denota el hecho de que podemos llegar de p a q mediante ε-movimientos, luego a, luego ε-movimientos. En el autómata del ejemplo 3.7 es cierto que $q_0\xrightarrow{0}q_1$, pues $q_0\in\delta(q_0,0)$ y $q_1\in\delta(q_0,\varepsilon)$, pero no es cierto que $q_1\in\delta(q_0,0)$.
2. Cuando un autómata no tiene ε-mov, entonces coinciden la función de transición y las transformaciones por símbolos de input, es decir si p,q son estados y a es un símbolo de input,

   $$p\in\delta(q,a)\quad\text{ si y sólo si }\quad q\xrightarrow{a}p.$$

   Por lo tanto, la notación $p\xrightarrow{\alpha}q$ para NFA con ε-mov, es consistente con la misma notación para DFA.
3. Es claro, en forma intuitiva, que si $\alpha,\beta\in\Sigma^*$, entonces para $p,q\in Q$,

   $$p\xrightarrow{\alpha\beta}q\quad\text{ si y sólo si existe } r\in Q \text{ tal que }\quad p\xrightarrow{\alpha}r\xrightarrow{\beta}q. \tag{1}$$

4. Sea $M'=(Q,\Sigma,\delta',q_0,F)$ con δ' definida de la siguiente manera:

   $$q\in\delta'(p,a)\quad\text{ sii }\quad p\xrightarrow{a}q.$$

   Entonces M' resulta ser un NFA con ε-mov que acepta el mismo lenguaje que M. La demostración se deja como ejercicio.

5. Debido a la observación anterior, en gran parte de las situaciones que necesitemos usar transiciones podremos tomarnos la licencia de usar la notación

$$p \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n \xrightarrow{a_n} q,$$

para indicar que hay un camino de flechas con etiquetas $a_0, \ldots, a_n$ que pasa por los estados $q_1, \ldots, q_n$ y que va de p a q. Dicho más formalmente:

$$q_1 \in \delta(p, a_0),\ q_2 \in \delta(q_1, a_1),\ \ldots q_n \in \delta(q_{n-1}, a_{n-1}),\ q \in \delta(q_n, a_n).$$

Repetimos la

Definición 3.10. Una cadena α es *aceptada* por el autómata si α transforma el estado inicial en uno final. Finalmente el *lenguaje aceptado por M* es el conjunto $L(M)$ de todas las cadenas aceptadas. Simbólicamente:

$$L(M) = \{\alpha \in \Sigma^* \mid q_0 \xrightarrow{\alpha} p, \text{ para algún } p \in F\}.$$

El siguiente Teorema fue anunciado al comienzo de la sección:

Teorema 3.1. *Los lenguajes aceptados por los autómatas finitos determinísticos son los mismos que los aceptados por los autómatas finitos no determinísticos con* ε*-movimientos. Más precisamente, dado un DFA (NFA con* ε*-movimientos) M, existe un NFA con* ε*-movimientos (resp. DFA) M′, tal que* $L(M) = L(M')$.

Demostración. Si $M = (Q, \Sigma, \delta, q_0, F)$ un DFA, entonces sea $M' = (Q, \Sigma, \delta', q_0, F)$ el NFA definido por $\delta'(q,a) = \{\delta(q,a)\}$ para $q \in Q$ y $a \in \Sigma$. Es trivial comprobar, entonces, que $L(M) = L(M')$.

Veremos ahora que, dado $M = (Q, \Sigma, \delta, q_0, F)$ un NFA con ε-mov, podemos construir $M' = (Q', \Sigma, \delta', q_0', F')$ un DFA, tal que $L(M) = L(M')$.

Sea $q \in M$, definamos $[q] = \{p \in Q \mid q \xrightarrow{\varepsilon} p\} \subseteq Q$. Sean $q_1, \ldots, q_i$ elementos de Q, entonces denotemos

$$[q_1, \ldots, q_i] = \cup_{j=1}^{i} [q_j].$$

Definamos M' de la siguiente manera: $Q' = \{[q_1, \ldots, q_i] \mid q_1, \ldots, q_i \in Q\}$. El nuevo conjunto de estados finales, F', es el conjunto de estados que contienen un estado que se transforma por ε en un estado final de M. El estado inicial será $q_0' = [q_0]$. Finalmente, definimos $\delta'([q_1, \ldots, q_i], a)$ igual al conjunto $\{p \in Q \mid \text{existe } q_s \text{ tal que } q_s \xrightarrow{a} p\}$. Es decir, un $a \in \Sigma$ transforma $q' \in Q'$ en $p' \in Q'$, si el conjunto p' está formado por todos los elementos de Q que son los transformados por a de los elementos de q'. Resumiendo, si $q' \in Q'$ y $a \in \Sigma$:

$$\begin{aligned}
Q' &= \{[q_1, \ldots, q_i] \mid q_1, \ldots, q_i \in Q\} \subseteq \mathcal{P}(Q) \\
q_0' &= [q_0] \\
F' &= \{[q_1, \ldots, q_i] \mid \text{ existe } k \text{ y } p \in F \text{ tal que } q_k \xrightarrow{\varepsilon} p\} \\
\delta(q', a) &= \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{a} p\} \quad \text{o equivalentemente} \\
q' &\xrightarrow{a} \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{a} p\}.
\end{aligned}$$

No es difícil comprobar que $M' = (Q', \Sigma, \delta', q_0', F')$ es un autómata finito determinístico.
No es difícil comprobar que si $q' \in Q'$ entonces

$$q' = \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\varepsilon} p\}. \tag{2}$$

Esto se debe a que como $q' = [q_1, \ldots, q_i]$, por definición

$$\begin{aligned} q' &= \cup_{j=1}^{i} [q_j] = \cup_{j=1}^{i} \{p \in Q \mid q_j \xrightarrow{\varepsilon} p\} \\ &= \{p \in Q \mid \exists q_j \text{ tal que } q_j \xrightarrow{\varepsilon} p\} \subset \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\varepsilon} p\}. \end{aligned}$$

Para demostrar la otra inclusión, consideremos p tal que $q \xrightarrow{\varepsilon} p$ con $q \in q'$. Como $q \in q'$ existe q_j tal que $q_j \xrightarrow{\varepsilon} q$, luego $q_j \xrightarrow{\varepsilon} q \xrightarrow{\varepsilon} p$, de lo cual se deduce que $q_j \xrightarrow{\varepsilon} p$ y por lo tanto $p \in \{p \in Q \mid \exists q_j \text{ tal que } q_j \xrightarrow{\varepsilon} p\}$.

Veamos ahora que si $\alpha \in \Sigma^*$, q' en Q', entonces:

$$q' \xrightarrow{\alpha} \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\alpha} p\}.$$

Supongamos que $q' \xrightarrow{\alpha} p'$ y veamos que $p' = \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\alpha} p\}$: hagamos inducción sobre $|\alpha|$. Cuando $|\alpha| = 0$, es decir cuando $\alpha = \varepsilon$, tenemos que $q' \xrightarrow{\varepsilon} q'$ y el párrafo anterior demuestra el resultado. Cuando $|\alpha| = 1$ el resultado se deduce trivialmente por definición de transición. Si $|\alpha| > 1$, entonces $\alpha = \beta a$, con $|\beta| \geq 1$ y $a \in \Sigma$. Por (**??**) de la observación 2.2 sabemos que existe $r' \in Q'$ tal que $q' \xrightarrow{\beta} r' \xrightarrow{a} p'$. Por hipótesis inductiva y el caso de longitud 1 tenemos que

$$\begin{aligned} q' \xrightarrow{\beta} r' &= \{r \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\beta} r\}, \\ r' \xrightarrow{a} p' &= \{p \in Q \mid \exists r \in r' \text{ tal que } r \xrightarrow{a} p\}. \end{aligned}$$

Si $p \in p'$, existe $r \in r'$ tal que $r \xrightarrow{a} p$, como $r \in r'$, existe $q \in q'$, tal que $q \xrightarrow{\beta} r$. Es decir que $q \xrightarrow{\beta a} p$ o equivalentemente $q \xrightarrow{\alpha} p$. Por lo tanto está probado $p' \subseteq \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\alpha} p\}$. Por otro lado, si $p \in \{p \in Q \mid \exists q \in q' \text{ tal que } q \xrightarrow{\alpha} p\}$, existe $q \in q'$ tal que $q \xrightarrow{\alpha} p$, luego existe $r \in Q$ tal que $q \xrightarrow{\beta} r \xrightarrow{a} p$, por lo tanto $r \in r'$ y $p \in p'$. Esto prueba la otra inclusión y por lo tanto la igualdad deseada.

La aplicación directa del resultado anterior a q'_0, nos dice que dada $\alpha \in \Sigma^*$, entonces

$$q'_0 \xrightarrow{\alpha} \{p \in Q \mid q_0 \xrightarrow{\alpha} p\}. \tag{3}$$

Ahora bien, sea $\alpha \in L(M')$, es decir $q'_0 \xrightarrow{\alpha} p'$, con $p' \in F'$. Como p' es final de M', entonces existe p en p' que también pertenece a F, luego por (3) obtenemos que $q_0 \xrightarrow{\alpha} p$, es decir $\alpha \in L(M)$. Recíprocamente, sea $\alpha \in L(M)$, es decir existe $p \in F$ tal que $q_0 \xrightarrow{\alpha} p$, por lo tanto (usando nuevamente (3)) $p \in p'$ con $q'_0 \xrightarrow{\alpha} p'$. De esto se deduce que $p' \in F'$ y $\alpha \in L(M')$.

Finalmente, el párrafo anterior prueba que $L(M) = L(M')$.

□

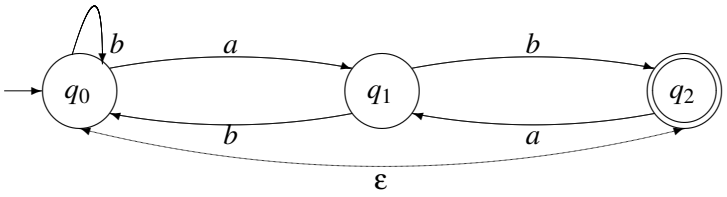
Ejemplo 3.13. Sea M el NFA con ε-mov con estados q_0, q_1, un solo símbolo de input a, estado inicial q_0, estado final q_1 y las siguientes leyes de transición:

$$\delta(q_0, a) = \{q_0\}, \qquad \delta(q_0, \varepsilon) = \{q_1\}.$$

Encontremos un autómata determinístico con el mismo lenguaje que M. Por la demostración del teorema el nuevo autómata, M', tendrá estados $\emptyset, [q_0]$. Observar que $[q_0] = [q_1] = [q_0, q_1] = \{q_0, q_1\}$, pues $q_0 \xrightarrow{\varepsilon} q_1$. El estado final e inicial es entonces $[q_0]$. Por definición, del estado $\emptyset$ no sale ninguna flecha y $\delta([q_0], \varepsilon) = [q_0]$, $\delta([q_0], a) = [q_0]$. Claramente, el lenguaje aceptado por ambos autómatas es el de todas las cadenas de a's.

Veamos un ejemplo menos trivial.

Ejemplo 3.14. Sea $M = (Q, \Sigma, \delta, q_0, F)$ un NFA con ε-movimientos definido por el siguiente diagrama de transición.



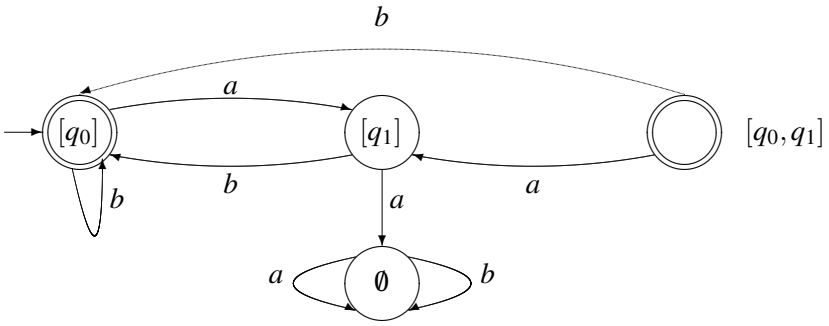
Encontremos un DFA que acepte el mismo lenguaje. Debemos primero establecer los estados del nuevo autómata:

$$\begin{aligned} [q_0] &= \{p \in Q \mid q_0 \xrightarrow{\varepsilon} p\} = \{q_0, q_2\} \\ [q_1] &= \{p \in Q \mid q_1 \xrightarrow{\varepsilon} p\} = \{q_1\} \\ [q_0, q_1] &= [q_0] \cup [q_1] = \{q_0, q_1, q_2\} \end{aligned}$$

Otro estado, siempre presente es $\emptyset$. Observemos que cualquier otro posible estado es uno de los ya listados más arriba, por ejemplo, $[q_2] = [q_0]$ y $[q_1, q_2] = [q_1] \cup [q_2] = [q_0, q_1]$. Establezcamos ahora las transiciones.

$$\begin{aligned} [q_0] &\xrightarrow{a} \{p \in Q \mid \exists q \in [q_0] \text{ tal que } q \xrightarrow{a} p\} = \\ &\qquad = \{p \in Q \mid q_0 \xrightarrow{a} p\} \cup \{p \in Q \mid q_2 \xrightarrow{a} p\} = \{q_1\} \cup \{q_1\} = [q_1] \\ [q_0] &\xrightarrow{b} \{p \in Q \mid q_0 \xrightarrow{b} p\} \cup \{p \in Q \mid q_2 \xrightarrow{b} p\} = \{q_0, q_2\} \cup \emptyset = [q_0] \\ [q_1] &\xrightarrow{a} \{p \in Q \mid q_1 \xrightarrow{a} p\} = \emptyset \\ [q_1] &\xrightarrow{b} \{p \in Q \mid q_1 \xrightarrow{b} p\} = \{q_0, q_2\} = [q_0] \\ [q_0, q_1] &\xrightarrow{a} \{p \in Q \mid \exists q \in [q_0, q_1] \text{ tal que } q \xrightarrow{a} p\} = [q_1] \cup \emptyset = [q_1] \\ [q_0, q_1] &\xrightarrow{b} [q_0] \cup [q_0] = [q_0] \end{aligned}$$

Por definición, es claro que toda transición que sale de $\emptyset$ vuelve a $\emptyset$. Finalmente, el estado inicial y final es $[q_0]$. El diagrama de transición del autómata recién construido es:



3.3 Expresiones regulares

Los lenguajes aceptados por los autómatas finitos se pueden describir fácilmente por expresiones simples llamadas expresiones regulares. En esta sección introduciremos las operaciones de concatenación y clausura sobre conjuntos de cadenas, definiremos expresiones regulares y daremos la demostración de que una expresión regular define un lenguaje que puede ser descrito por un autómata finito. La recíproca de este resultado también es cierta y veremos su demostración en la subsección final.

Definición 3.11. Sea Σ un conjunto finito de símbolos y sean L, L_1 y L_2 conjuntos de cadenas de Σ^*. La *concatenación* de L_1 y L_2, denotada L_1L_2 es un conjunto formado por cadenas de L_1 seguidas por cadenas de L_2, es decir: $L_1L_2 = \{xy \mid x \text{ está en } L_1 \text{ e } y \text{ está en } L_2\}$. Definamos $L^0 = \{\varepsilon\}$ y $L^i = LL^{i-1}$ para $i \geq 1$. La *clausura de Kleene* (o sólo *clausura*) de L, denotada L^*, es el conjunto

$$L^* = \cup_{i=0}^{\infty} L^i$$

y la clausura positiva de L, denotada L^+, es el conjunto

$$L^+ = \cup_{i=1}^{\infty} L^i.$$

L^* denota las cadenas construidas por concatenación de un número arbitrario de cadenas de L. L^+ es igual pero sin poner L^0. Nótese que L^+ contiene ε si y sólo si L lo contiene. Recordemos que en la Sección **??** definimos Σ^*, y observemos que ese conjunto coincide con el Σ^* que surge de la definición anterior, considerando a Σ como un lenguaje donde las cadenas son los símbolos del alfabeto.

Ejemplo 3.15. Sea $L_1 = \{11, 0\}$ y $L_2 = \{001, 10\}$, entonces

$$L_1L_2 = \{11001, 1110, 0001, 010\}.$$

Por otro lado,

$$L_1^* = \{11, 0\}^* = \{\varepsilon, 11, 0, 1111, 110, 011, 00, 111111, 11110, 11011, 1100, \ldots\}.$$

L_1^+ es L_1^* menos el ε.

Definición 3.12. Sea Σ un alfabeto. Las *expresiones regulares* sobre Σ y los *conjuntos* que ellas denotan se definen recursivamente de la siguiente manera:

1. $\emptyset$ es una expresión regular y denota el conjunto vacío.
2. ε es un expresión regular y denota el conjunto $\{\varepsilon\}$.
3. Para cada a en Σ, a es una expresión regular que denota el conjunto $\{a\}$.
4. Si r y s son expresiones regulares que denotan los conjuntos R y S, respectivamente, entonces $(r+s)$, (rs) y (r^*) son expresiones regulares que denotan los conjuntos $R \cup S$, RS y R^*, respectivamente.

Si r es una expresión regular escribiremos $L(r)$ al conjunto (o lenguaje) denotado por r.

Cuando escribimos expresiones regulares podemos omitir muchos paréntesis si asumimos que $*$ tiene más alta precedencia que la concatenación o $+$, y la concatenación tiene más alta precedencia que $+$. Por ejemplo $((0(1^*))+0)$ puede ser escrita 01^*+0. Finalmente, si r es una expresión regular denotemos r^i a la expresión $rr\cdots r$ (i veces).

Debemos tener mucho cuidado en no confundir cadenas con expresiones regulares y para ello debemos aclarar debidamente a qué nos estamos refiriendo. Por ejemplo la expresión regular ab denota un conjunto cuyo único elemento es la cadena ab.

Ejemplo 3.16. Los siguientes son ejemplo de expresiones regulares:

1. 00 es una expresión regular que representa $\{00\}$.
2. $(0+1)^*$ denota todas las cadenas de 0's y 1's.
3. $(0+1)^*00(0+1)^*$ denota todas las cadenas de 0's y 1's con al menos dos 0's consecutivos.
4. $(1+10)^*$ denota todas las cadenas de 0's y 1's que comienzan con 1 y no tienen dos 0's consecutivos. También pertenece a este conjunto la cadena vacía.

   Demostración. Probemos por inducción que $(1+10)^i$ no tiene dos 0's consecutivos. El caso $i=0$ es trivial, y si suponemos que $(1+10)^{i-1}$ no tiene dos 0's consecutivos, entonces, es claro que $(1+10)^i = (1+10)^{i-1}(1+10)$ no tiene dos 0's consecutivos, pues en el final las palabras son de la forma $\cdots\mathbf{1}1$ o $\cdots\mathbf{1}10$ o $\cdots\mathbf{10}1$ o $\cdots\mathbf{10}10$, donde las letras en negrita denotan palabras de $(1+10)^{i-1}$. De esta forma probamos que $(1+10)^i$ no tiene dos 0's consecutivos para todo i y por lo tanto $(1+10)^*$ no tiene dos 0's consecutivos. Por otro lado, dada cualquier cadena que comience con 1 y no tenga dos 0's consecutivos, se puede hacer una partición de la cadena de tal forma que si un 1 no es seguido de un 0, se toma el 1 (que pertenece a $(1+10)$) y si un 1 es seguido de un 0, se toma el 10 (que pertenece a $(1+10)$). Por ejemplo, 11010111 es particionado $1-10-10-1-1-1$ que pertenece a $(1+10)^6$. □
5. Basado en lo anterior, es claro que $(0+\varepsilon)(1+10)^*$ denota las cadenas de 0's y 1's que no tienen dos 0's seguidos.
6. $(0+1)^*011$ denota las cadenas de 0's y 1's que terminan en 001.
7. $0^*1^*2^*$ denota las cadenas que están formadas por cierta cantidad de 0's, luego cierta cantidad de 1's y luego cierta cantidad de 2's. Observar que este es lenguaje del autómata definido en el ejemplo 3.7.

Veremos ahora como a partir de una expresión regular r podemos construir un autómata no determinístico M que defina el mismo lenguaje, es decir tal que $L(r) = L(M)$. En realidad lo que construiremos son autómatas no determinísticos con un sólo estado final. Para las expresiones regulares ε, $\emptyset$ o a con $a \in \Sigma$ los autómatas no determinísticos de la Fig. 10 definen el lenguaje correspondiente:

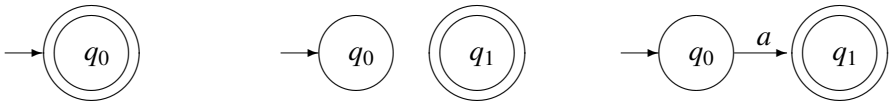


Figura 10:

A un autómata M con estado inicial q y estado final f lo dibujaremos de la siguiente manera (Fig. 11):

Ahora haremos lo siguiente: dadas expresiones regulares r_1 y r_2 a las cuales ya les hayamos asociado los autómatas M_1, M_2 correspondientes, como en la Fig. 12 encontraremos los autómatas correspondientes a $r_1 + r_2$, $r_1 r_2$ y r_1^*. Dicho de otra manera: si L_1 es el lenguaje de M_1 y L_2 es el lenguaje de M_2, entonces encontraremos autómatas correspondientes a $L_1 \cup L_2$, $L_1 L_2$ y L_1^*. El autómata correspondiente a $r_1 + r_2$ será el de la Fig. 13.

El autómata correspondiente a $r_1 r_2$ será el de la Fig. 14.

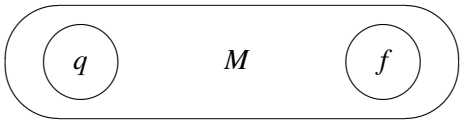


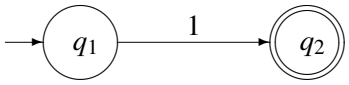
Figura 11:

Figura 12: Los autómatas de r_1 y r_2.

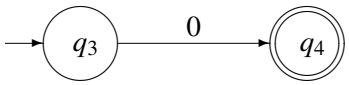
El autómata correspondiente a r_1^* será el de la Fig. 15
De lo anterior se deduce

Teorema 3.2. *Sea L una lenguaje denotado por la expresión regular r. Entonces existe M un DFA con* ε*-mov tal que* $L = L(M)$.

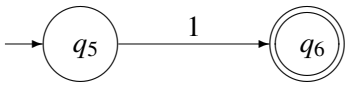
Ejemplo 3.17. Construyamos un autómata para la expresión regular $01^* + 1$. Por lo dicho anteriormente (respecto a los paréntesis), esta expresión es en realidad $((0(1^*)) + 1)$, es decir es de la forma $r_1 + r_2$, donde $r_1 = 01^*$ y $r_2 = 1$. El autómata para r_2 es fácil



Podemos expresar r_1 como $r_3 r_4$, donde $r_3 = 0$ y $r_4 = 1^*$. El autómata de r_3, también es fácil:



Ahora bien, r_4 es r_5^*, con $r_5 = 1$. Un autómata para r_5 es



Observemos que para poder construir el autómata correspondiente a la expresión regular necesitamos distinguir los estados de r_2 y r_5, aunque representen la misma expresión. Basándonos en las explicaciones anteriores un autómata para r_4 será

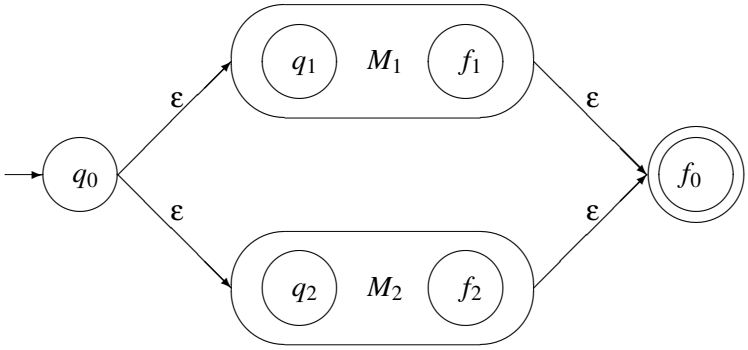


Figura 13: El autómata de $r_1 + r_2$.

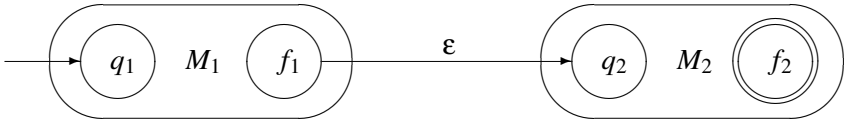
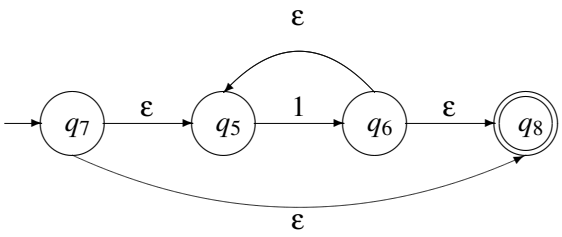
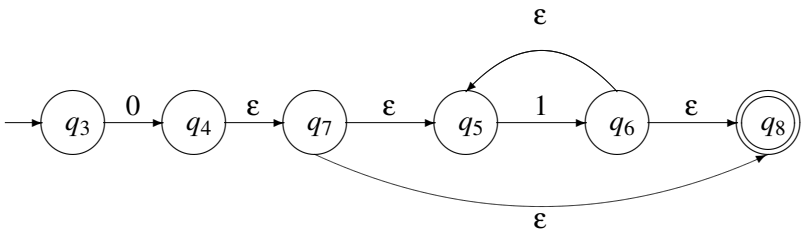


Figura 14: El autómata de $r_1 r_2$.



Luego el autómata correspondiente a la expresión regular $r_1 = 01^*$ será



Finalmente el autómata correspondiente a $01^* + 1$ será el de la Fig. 16.

3.3.1 Teorema de Kleene

Estudiemos el problema, un poco más complicado, de construir una expresión regular que denote el lenguaje de un autómata dado. Esta construcción se basa en la idea de encontrar, en forma algorítmica, una expresión regular que involucra resolver el problema para un autómata con menos estados. Repitiendo el proceso un número conveniente de veces lograremos la expresión regular buscada. Para

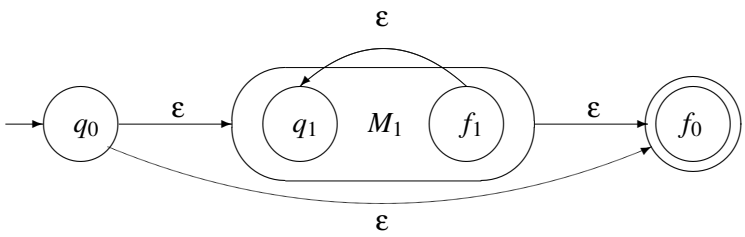


Figura 15: El autómata de r_1^*.

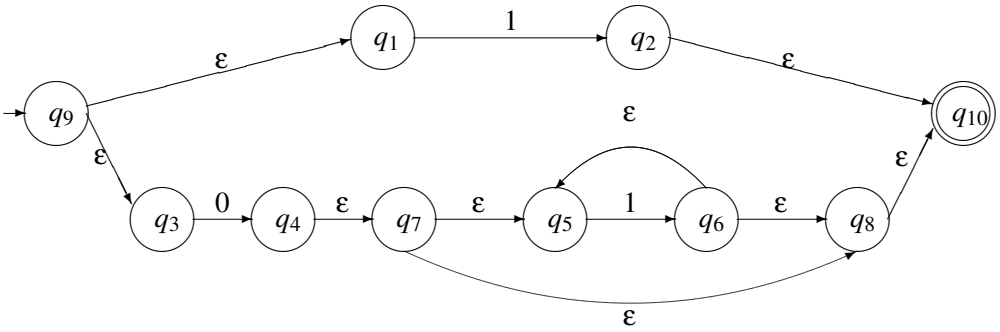


Figura 16: Un autómata correspondiente a la expresión regular $01^* + 1$.

describir el procedimiento, comenzaremos con el caso más simple en el cual nuestro autómata M tiene un solo estado final, q_n y su estado inicial es q_0. Es claro que $L(M) = L_{0n}$, el lenguaje de las cadenas que comienzan en q_0 y llegan a q_n. Llamemos I_0 al lenguaje formado por las cadenas que salen de q_0 y vuelven a q_0 *sin pasar nuevamente por él.* Si el estado inicial es el estado final, (es decir, $n = 0$) entonces claramente $L_{0n} = I_0^*$, puesto que con repetir caminos que salgan de q_0 y vuelvan a él obtengo todas las palabras aceptadas. Si $n \neq 0$, definamos F_{0n} como el lenguaje de las cadenas que salen de q_0 y llegan a q_n *sin pasar por* q_0. Entonces, por un razonamiento similar, $L_{0n} = I_0^* F_{0n}$.

Debemos ahora explicitar quiénes son I_0 y F_{0n}. Los elementos de I_0 son cadenas de dos formas:

1. $a\beta_0 b$, donde $a, b \in \Sigma \cup \{\varepsilon\}$ y β_0 es una cadena que parte de un estado p y tal que $q_0 \xrightarrow{a} q_i \xrightarrow{\beta_0} q_j \xrightarrow{b} q_0$ con p y q distintos de q_0 y además el camino que recorre β_0 no pasa por q_0; o bien
2. $c \in \Sigma \cup \{\varepsilon\}$, que haga un bucle de q_0 en sí mismo.

Si consideramos en el primer caso el autómata M' que se obtiene de M sacando q_0 y haciendo que q_i sea estado inicial y q_j estado final, entonces β_0 es una cadena aceptada por M'. En este caso, $L(M') = L_{ij}$, el lenguaje de las cadenas que salen de q_i y llegan a q_j sin pasar nunca por q_0, y obtenemos:

$$I_0 = aL_{ij}b + \cdots + c + \ldots$$

donde se suman todas las posibilidades para a, b, i, j tales que $q_0 \xrightarrow{a} q_i$, $q_j \xrightarrow{b} 0$, con q_0 distinto de q_i y q_j, los c tales que $q_0 \xrightarrow{c} q_0$, y en L_{ij} no consideramos q_0. Es decir, las cadenas que parten y llegan a q_0 se pueden ver como la concatenación de cadenas formadas por ciertos símbolos de entrada, con lenguajes aceptados por autómatas más chicos.

De igual modo, si q_0 no es el estado final, los elementos de F_{0n} son las cadenas que parten de q_0 y llegan al estado final q_n sin pasar por q_0. En este caso las cadenas son de la forma $a\beta$ con

$q_0 \xrightarrow{a} q_j \xrightarrow{\beta} q_n$, donde $a \in \Sigma \cup \{\varepsilon\}$. Luego esta cadena es la concatenación de un símbolo de entrada y una cadena aceptada por un autómata más chico:

$$F_{0n} = aL_{jn} + \ldots$$

donde a, j se mueven entre todas las opciones tales que $q_0 \xrightarrow{a} q_j \xrightarrow{\beta} q_n$, $j \neq 0$ y en L_{ij} (igual que arriba) no consideramos a q_0. Dado que el problema se va reduciendo a autómatas mas chicos, este algoritmo termina.

El caso de autómatas M con múltiples estados finales se trata similarmente, escribiendo

$$L(M) = aL_{jk} + \ldots,$$

donde a, j, k se mueven entre las opciones tales que $q_0 \xrightarrow{a} q_j \xrightarrow{\beta} q_k$ con q_k un estado final.

Ejemplo 3.18. Encontremos una expresión regular para el lenguaje aceptado por el autómata M descrito mediante el diagrama de transición de la Figura 17.

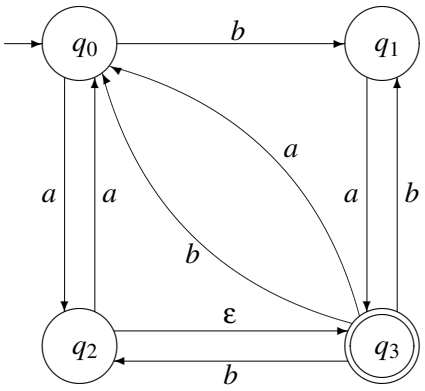


Figura 17: El autómata M

Es claro que $L(M) = L_{03}$, el lenguaje de las cadenas que comienzan en q_0 y llegan a q_3. Llamemos I_0 al lenguaje formado por las cadenas que salen de q_0 y vuelven a q_0 *sin pasar nuevamente por él* y F_{03} al lenguaje de las cadenas que salen de q_0 y llegan a q_3 *sin pasar por* q_0. Entonces, $L_{03} = I_0^* F_{03}$.

Analicemos ahora quiénes son I_0 y $F_{0,3}$.

Las formas de salir de q_0 y volver a sí mismo sin pasar nuevamente por él se pueden escribir así:

$$I_0 = bL_{12}a + bL_{13}a + bL_{13}b + aL_{23}a + aL_{23}b + aL_{22}a,$$

$$I_0 = b(L_{12}a + L_{13}(a+b)) + a(L_{23}(a+b) + L_{22}a),$$

donde L_{12} es el lenguaje aceptado por el autómata M_{12} que consta de los estados $\{q_1, q_2, q_3\}$ (hemos eliminado q_0), y que tiene por estado inicial a q_1 y final a q_2. De manera similar se definen $L_{13} = L(M_{13})$, $L_{23} = L(M_{23})$ (ver Figura 19) y L_{22} (ver Figura 18).

Las maneras de salir de q_0 y llegar al estado final q_3 sin pasar por q_0 son:

$$F_{03} = bL_{13} + aL_{23}.$$

Tenemos, como antes $L_{12} = I_1^* F_{12}$, salvo que ahora hemos eliminado q_0 de nuestras consideraciones. Es decir,

$$I_1 = aL_{33}b, \qquad F_{12} = aL_{33},$$

son las formas de ir de q_1 a q_1 y de q_1 a q_3, respectivamente, sin pasar nuevamente por q_1 (y sin utilizar q_0). Ahora, en nuestro L_{33}, no utilizaremos ni q_0 ni q_1.

> Como nuestro estado inicial es igual al estado final (q_3), tenemos
>
> $$L_{33} = I_3^*.$$
>
> En I_3 no usaremos ni q_0 ni q_1. Entonces obtenemos:
>
> $$I_3 = bL_{22}\varepsilon = b,$$
>
> con lo que
>
> $$L_{33} = b^*.$$

Ya podemos calcular los anteriores $I_1 = aL_{33}b = ab^*b$ y $F_{12} = aL_{33} = ab^*$. Luego obtenemos

$$L_{12} = (ab^*b)^*ab^*.$$

Ahora nos toca ver $L_{13} = I_1^* F_{13}$, eliminando q_0.

$$I_1 = ab^*b, \qquad F_{13} = aL_{33} = ab^*,$$

dado que, como antes, en L_{33} eliminamos q_0 y q_1. Entonces $L_{13} = (ab^*b)^*ab^*$.

Observemos que $L_{23} = I_2^* F_{23}$, eliminando q_0.

$$I_2 = \varepsilon L_{33}b = L_{33}b, \qquad F_{23} = \varepsilon L_{33} = L_{33},$$

donde en L_{33} se eliminan q_0 y q_2.

> Veamos L_{33} con q_0 y q_2 eliminados. Como antes (estado inicial es el final) tenemos
>
> $$L_{33} = I_3^*.$$
>
> En I_3 no usaremos ni q_0 ni q_2. Entonces obtenemos:
>
> $$I_3 = bL_{11}a = ba$$
>
> con lo que
>
> $$L_{33} = (ba)^*.$$

Obtenemos entonces

$$I_2 = (ba)^*b, \qquad F_{23} = (ba)^*,$$

y luego $L_{23} = ((ba)^*b)^*(ba)^*$.

Sólo nos falta $L_{22} = I_2^*$ (Figura 18). Tenemos

$$I_2 = \varepsilon L_{33} b = L_{33} b = (ba)^* b,$$

pues es el mismo L_{33} considerado anteriomente. Entonces

$$L_{22} = ((ba)^* b)^*.$$

Por último,

$$I_0 = b(L_{12}a + L_{13}(a+b)) + a(L_{23}(a+b) + L_{22}a),$$

$$I_0 = b((ab^*b)^* ab^* a + (ab^*b)^* ab^*(a+b)) + a(((ba)^*b)^*(ba)^*(a+b) + ((ba)^*b)^* a).$$

$$F_{03} = bL_{13} + aL_{23},$$

$$F_{03} = b(ab^*b)^* ab^* + a((ba)^*b)^*(ba)^*,$$

y nuestro resultado final es

$$L = L_{03} = I_0^*\, F_{03}.$$

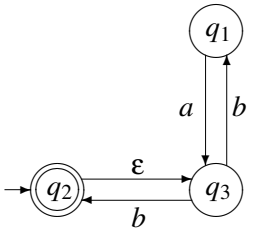


Figura 18: El autómata M_{22}

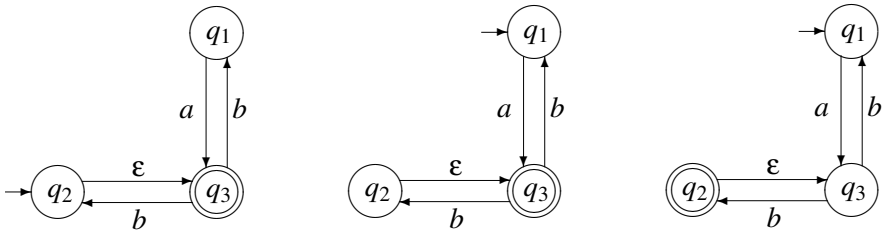


Figura 19: Los autómatas M_{23}, M_{13} y M_{12}, respectivamente

Teorema 3.3 (Teorema de Kleene)**.** *Sea M un NFA con ε-mov, entonces, existe una expresión regular r tal que $L(M)$ es el lenguaje denotado por r.*

Probaremos este teorema viendo que, dado un NFA M, el algoritmo descripto anteriormente devuelve una expresión regular que caracteriza el lenguaje aceptado por M. Para ello será necesario formalizar las definiciones de L_{nm}, I_n, F_{nm} y el proceso de "eliminar estados".

Supongamos que $M = (Q, \Sigma, \delta, q_0, F)$. Supondremos que existe un único estado final: $F = \{q_f\}$, con $0 \leq f \leq r$. Sea $Q = \{q_0, \ldots, q_r\}$ y $\Sigma = \{c_1, \ldots, c_u\}$.

Dados $0 \leq n, m \leq r$, $R \subseteq Q$, definamos recursivamente las siguientes expresiones regulares:

$$\begin{array}{rl} L_{nm}(R) & = \emptyset \qquad \text{si } n \text{ ó } m \text{ no están en } R. \\ L_{nn}(R) & = I_n(R)^* \\ L_{nm}(R) & = I_n(R)^* F_{nm}(R) \qquad \text{si } n \neq m. \\ I_n(R) & = \sum_{q_n \xrightarrow{a} q_t,\, q_s \xrightarrow{b} q_n} a L_{ts}(R \setminus \{q_n\}) b \;\; + \sum_{q_n \xrightarrow{c} q_n} c \\ F_{nm}(R) & = \sum_{q_n \xrightarrow{a} q_t} a L_{tm}(R \setminus \{q_n\}), \end{array}$$

donde $a, b, c \in \Sigma \cup \{\varepsilon\}$. En particular,

$$L_{nn}(\{q_n\}) = I_n(\{q_n\})^* = \left(\sum_{q_n \xrightarrow{c} q_n} c \right)^*.$$

En esencia, en las expresiones $L_{nm}(R)$ sólo consideramos "habilitados" los estados del conjunto R.

Por otro lado, sea $M_{nm}(R) = (R, \Sigma, \delta \mid R, q_n, \{q_m\})$ el autómata que resulta de M luego eliminar todos los estados que no están en R y sus transiciones, y estipulando que el estado inicial es q_n y el único estado final es q_m. En caso de que q_n ó q_m no estén en R, $M_{nm}(R)$ será un autómata vacío con lenguaje $\emptyset$.

Lema 3.4. *Toda cadena α representada por las expresiones regulares de arriba son aceptadas por los respectivos autómatas. Es decir: para todo $R \subseteq Q$, $L_{nm}(R) \subseteq L(M_{nm}(R))$.*

Demostración. Tenemos dos casos a probar, correspondientes a la definición de $L_{nm}(R)$:

(1) Si $q_n \in R$ y $\alpha \in I_n(R)^k$ entonces $\alpha \in L(M_{nn}(R))$.
(2) Si $n \neq m$, $q_n, q_m \in R$ y $\alpha \in I_n(R)^k F_{nm}(R)$ entonces $\alpha \in L(M_{nm}(R))$.

Procederemos simultáneamente por inducción en k y en $|\, R \,|$.

El caso base para R es el conjunto vacío, para el cual los antecedentes de (1) y (2) son falsos. Más aún, por las definiciones tenemos que si alguno de n ó m falta en R, $L_{nm}(R) = L(M_{nm}(R)) = \emptyset$. Luego, podemos considerar de ahora en adelante que $n, m \in R$.

$\boxed{k=0}$ En el caso (1) debe ser $\alpha = \varepsilon$ y claramente $\varepsilon \in L(M_{nn}(R))$ al ser q_n estado inicial y final.
En el caso (2), $\alpha \in F_{nm}(R) = \sum_{q_n \xrightarrow{a} q_t} a L_{tm}(R \setminus \{q_n\})$. Luego existen a, t, α' tales que

$$q_n \xrightarrow{a} q_t, \quad \alpha' \in L_{tm}(R \setminus \{q_n\}), \quad \alpha = a\alpha'.$$

Por hipótesis inductiva (eliminamos un estado) $\alpha' \in L(M_{tm}(R \setminus \{q_n\}))$ y luego existe una sucesión

$$q_t = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_l} s_{l+1} = q_m$$

con $s_j \in R \setminus \{q_n\}$ y $\alpha' = a_1 a_2 \ldots a_l$. Como $q_n \xrightarrow{a} q_t$ entonces $\alpha \in L(M_{nm}(R))$.

$\boxed{k+1}$ Caso (1): es muy similar al (2), lo dejamos como ejercicio.
Caso (2): Supongamos que $\alpha \in I_n(R)^{k+1} F_{nm}(R)$. Luego, hay dos opciones (para cada uno de los tipos de sumandos en la definición de I_n):

- $\alpha = a_0\alpha'\beta'$ con $q_n \xrightarrow{a_0} q_n$, $\alpha' \in I_n(R)^k$ y $\beta' \in F_{nm}(R)$. Por hipótesis inductiva (disminuimos k), $\alpha'\beta' \in L(M_{nm}(R))$. Como antes, hay una sucesión

$$q_n = s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_l} s_{l+1} = q_m$$

tal que $\alpha'\beta' = a_1 a_2 \dots a_l$ y luego

$$q_n \xrightarrow{a_0} q_n \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \xrightarrow{a_l} s_{l+1} = q_m$$

muestra que $\alpha \in L(M_{nm}(R))$.

- $\alpha = a_0\gamma b_0\alpha'\beta'$ con $q_n \xrightarrow{a_0} s_0$, $t_0 \xrightarrow{b_0} q_n$, $\gamma \in L_{s_0t_0}(R \setminus \{q_n\})$, $\alpha' \in I_n(R)^k$ y $\beta' \in F_{nm}(R)$. Por hipótesis inductiva $\alpha'\beta' \in L(M_{nm}(R))$ (disminuimos k) y $\gamma \in L(M_{s_0t_0}(R \setminus \{q_n\}))$ (borramos q_n). Es decir

$$s_0 \xrightarrow{\gamma} t_0, \quad q_n \xrightarrow{\alpha'\beta'} q_m,$$

y luego

$$q_n \xrightarrow{a_0} s_0 \xrightarrow{\gamma} t_0 \xrightarrow{b_0} q_n \xrightarrow{\alpha'\beta'} q_m$$

muestra que $\alpha \in L(M_{nm}(R))$.

□

Lema 3.5. *Toda cadena α aceptada por $M_{nm}(R)$ está en el lenguaje representado por la expresión regular dada por el algoritmo. Es decir: para todo $R \subseteq Q$, $L(M_{nm}(R)) \subseteq L_{nm}(R)$.*

Demostración. En esta prueba es más conveniente por motivos técnicos considerar las sucesiones de transiciones que dan lugar a las cadenas aceptadas, que considerar las cadenas solamente. Probaremos, para toda sucesión de transiciones en M

$$q_n = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{l-1}} s_l = q_m$$

(donde $a_i \in \Sigma \cup \{\varepsilon\}$ y $s_i \in Q$), que:

si $s_i \in R$ para todo i entonces $a_1 \dots a_{l-1} \in L_{nm}(R)$,

por inducción en la longitud l de la sucesión. Llamemos $\alpha = a_1 \dots a_{l-1}$. Consideraremos un "caso (1)" cuando $n = m$ y un "caso (2)" cuando no sean iguales.

$\boxed{l = 0}$ Caso (1): si la cantidad de transiciones es cero, no nos movemos de q_n. La cadena representada es la vacía, y obviamente está en $I_n(R)^* = L_{nn}(R)$.

Caso (2): al ser $n \neq m$, l tiene que ser mayor a cero, así que este caso base es trivial.

$\boxed{l+1}$ Caso (1): si todos los s_i son iguales a q_n, entonces claramente

$$\alpha \in \left(\sum_{q_n \xrightarrow{c} q_n} c \right)^* \subseteq I_n(R)^* = L_{nn}(R)$$

Sino, tomemos $0 \leq j \leq h \leq l$ tales que s_i es distinto de q_n para todo i entre j y h, pero $s_{j-1} = s_{h+1} = q_n$. Luego existen α', β y γ (eventualmente vacías) tales que:

$$q_n = s_0 \xrightarrow{\alpha'} s_{j-1} \xrightarrow{a_{j-1}} s_j \xrightarrow{\beta} s_h \xrightarrow{a_h} s_{h+1} \xrightarrow{\gamma} s_l = q_n$$

Por hipótesis inductiva, tenemos $\alpha', \gamma \in L_{nn}(R) = I_n(R)^*$ y $\beta \in L_{jh}(R \setminus \{q_n\})$.[1] Luego

$$\alpha \in I_n(R)^* \left(a_{j-1} L_{jh}(R \setminus \{q_n\}) a_h\right) I_n(R)^* \subseteq I_n(R)^* I_n(R) I_n(R)^* = I_n(R)^*$$

Caso (2): sea s_h es el último s_j que es igual a q_n (notar que h debe ser menor que l puesto que $n \neq m$). Luego

$$q_n = s_0 \xrightarrow{\alpha'} s_h \xrightarrow{a_h} s_{h+1} \xrightarrow{\beta'} s_l = q_m$$

con $\alpha = \alpha' a_h \beta'$ con α' y β' eventualmente vacíos. Ahora, la sucesión de transiciones correspondiente a β' no contiene a q_n; por hipótesis inductiva tenemos que $\beta' \in L_{km}(R \setminus \{q_n\})$, donde suponemos sin pérdida de generalidad que $s_{h+1} = q_k$. Y también por hipótesis inductiva $\alpha' \in L_{nn}(R) = I_n(R)^*$. Entonces

$$\alpha \in I_n(R)^* a_h L_{km}(R \setminus \{q_n\}) \subseteq I_n(R)^* F_{nm}(R) = L_{nm}(R).$$

□

Prueba del Teorema de Kleene. Supongamos $M = (Q, \Sigma, \delta, q_0, F)$ donde $Q = \{q_0, \ldots, q_r\}$ y $F = \{q_k, \ldots, q_m\}$ con $0 \leq k \leq m \leq r$. Está claro que

$$L(M) = \bigcup_{k \leq i \leq m} L(M_{0i}(Q)),$$

dado que cada cadena aceptada termina obviamente en un solo estado final. Pero ahora, por los lemas anteriores tenemos

$$L(M) = \bigcup_{k \leq i \leq m} L_{0i}(Q)$$

y por último

$$L(M) = \sum_{k \leq i \leq m} L_{0i}(Q)$$

es la expresión regular buscada. □

4 Gramáticas y lenguajes

Los lenguajes pueden ser clasificados en dos grandes grupos: los lenguajes *naturales* (tal como el español, inglés, francés, etc.) y los lenguajes *formales* (como la matemática, la lógica, etc.). Los lenguajes naturales en general son más flexibles, lo cual los hace más difícil de caracterizar, mientras que los lenguajes formales poseen reglas de sintaxis y semántica más rígidas.

En la década del '50, el lingüista Noam Chomsky hace un gran aporte a las ciencias de la computación. En su trabajo *"Teoría de las Gramáticas Transformacionales"* no sólo introdujo herramientas para el estudio de lenguajes naturales, sino que también las bases para la definición de lenguajes formales, lo cual posteriormente tuvo consecuencias en la definición de lenguajes de programación.

Chomsky clasificó los lenguajes de acuerdo a una jerarquía de cuatro niveles, conteniendo cada uno a los siguientes. El lenguaje más general (de *tipo 0*) no posee restricción alguna. Este nivel engloba el conjunto de todos los lenguajes posibles. En el otro extremo de la clasificación los lenguajes de *tipo 3*, o *lenguajes regulares*, son los que presentan una estructura más sencilla. Además, gracias a la contribución realizada por Chomsky fue posible establecer una relación directa entre la teoría de lenguajes formales (el tema principal de esta sección) y la teoría de máquinas abstractas (estudiadas

[1] Aquí suponemos, por comodidad notacional, que $s_j = q_j$ y $s_h = q_h$.

en secciones anteriores). El principal resultado establece que existe una definición de equivalencia entre diferentes construcciones de ambas teorías.

En este capítulo introduciremos una construcción formal para la definición de lenguajes: las *gramáticas*. Introduciremos la jerarquía de Chomsky, pero nos centraremos en las gramáticas regulares y libres de contexto y los lenguajes que ellas describen (los lenguajes regulares y libres de contexto). Ya comenzamos el estudio de lenguajes regulares en las secciones anteriores, pero también nos centraremos en lenguajes libres de contexto ya que son de gran importancia en la definición de los lenguajes de programación, entre otra aplicaciones. Como un ejemplo, los lenguajes libres de contextos son útiles para describir expresiones aritméticas con paréntesis balanceados y para describir estructuras de bloque en los lenguajes de programación.

4.1 Definiciones básicas y ejemplos

En esta sección nos centraremos en la definición de una herramienta formal para describir lenguajes: las gramáticas. Las mismas además, serán de ayuda para caracterizar las diferentes categorías en la jerarquía de Chomsky.

Las siguientes definiciones serán de utilidad a la hora de introducir formalmente las gramáticas.

Definición 4.1. Sea Σ un alfabeto, $\alpha, \beta \in \Sigma^*$, decimos que (α, β) es una *producción*. También podemos denotarla como $\alpha \to \beta$, o $\alpha ::= \beta$.

Una gramática es un objeto formal para especificar de manera finita, el conjunto de cadenas que constituyen un lenguaje. Usamos las gramáticas para generar las distintas cadenas del lenguaje que define aplicando de manera sucesiva las reglas de producción que la conforman. Una gramática define la estructura de las frases y de las palabras de un lenguaje.

Definición 4.2. Una *gramática* es una 4-upla $\langle V, T, P, S \rangle$ tal que:

1. V es un conjunto de símbolos denominados *no terminales*.
2. T es un conjunto de símbolos denominados *terminales* ($V \cap T = \emptyset$).
3. P es el conjunto de *producciones* sobre $V \cup T$.
4. $S \in V$ es el *símbolo inicial* a partir del cual se aplican las derivaciones para obtener las cadenas del lenguaje.

Ejemplo 4.1. Definamos la siguiente gramática la cual genera un conjunto (muy acotado) de frases bien formadas del español.

$$
\begin{array}{lcl}
V & = & \{\langle oracion \rangle, \langle sujeto \rangle, \langle adjetivo \rangle, \langle predicado \rangle, \langle verbo \rangle\} \\
T & = & \{a, b, c, d, e, ..., z\} \\
S & = & \langle oracion \rangle \\
P & = & < \text{oración} > \to < \text{sujeto} >< \text{predicado} > \\
 & & < \text{sujeto} > \to < \text{sujeto} >< \text{adjetivo} > | \text{ el perro} \\
 & & < \text{adjetivo} > \to \text{pequeño} \mid \text{grande} \mid \text{bueno} \\
 & & < \text{predicado} > \to < \text{verbo} >< \text{adjetivo} > \\
 & & < \text{verbo} > \to \text{es}
\end{array}
$$

Nótese que se utiliza la abreviación $\alpha \to \beta_1 \mid \ldots \mid \beta_k$ cuando tenemos k producciones con el mismo símbolo a la izquierda.

Definición 4.3. Sea $G = (V, T, P, S)$ una gramática. Si $A \to \alpha$ es una producción y xAy es una cadena formada por variables y terminales, se dice que $x\alpha y$ *se deriva directamente* de xAy y se escribe

$$xAy \Rightarrow a\alpha y.$$

Si $\alpha_1, \ldots, \alpha_n$ cadenas y $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$ derivaciones directas, entonces decimos que α_n se deriva de α_1 y se escribe

$$\alpha_1 \Rightarrow \alpha_n.$$

La secuencia

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \cdots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$$

se llama *derivación* de α_1 a α_n. Por convención, cualquier cadena se deriva de si misma.

Finalmente, el *lenguaje generado* por G consiste de todas las cadenas de elementos de T que se derivan de S. Se denota $L(G)$.

Ejemplo 4.2. Ahora deseamos ver que frases se pueden armar con terminales partiendo de la variable $<$ oración $>$ y usando las reglas de producción. Por ejemplo:

$$\begin{aligned} <\text{ oración }> &\Rightarrow <\text{ sujeto }><\text{ predicado }> \\ &\Rightarrow <\text{ sujeto }><\text{ adjetivo }><\text{ predicado }> \\ &\Rightarrow \text{el perro} <\text{ adjetivo }><\text{ predicado }> \\ &\Rightarrow \text{el perro} <\text{ adjetivo }><\text{ verbo }><\text{ adjetivo }> \\ &\Rightarrow \text{el perro bueno} <\text{ verbo }><\text{ adjetivo }> \\ &\Rightarrow \text{el perro bueno es} <\text{ adjetivo }> \\ &\Rightarrow \text{el perro bueno es grande} \end{aligned}$$

Intuitivamente el símbolo $\Rightarrow$ denota la acción de derivar, es decir, reemplazar una variable por el lado derecho de una producción para la variable.

4.2 Clasificación de Chomsky

Clasificar los lenguajes consiste en identificar aquellos lenguajes que comparten alguna propiedad y reunirlos en una misma clase. Toda gramática genera un único lenguaje, pero distintas gramáticas pueden definir el mismo lenguaje. Por lo tanto, podríamos clasificar las gramáticas según el lenguaje que generan. Es decir dos gramáticas pertenecerán a la misma clase si generan el mismo lenguaje. Sin embargo existe un problema con esta clasificación: no es decidible decidir si dos gramáticas generan el mismo lenguaje. Intuitivamente:

No existe ningún algoritmo que dadas dos gramáticas retorne si las dos definen el mismo lenguaje.

Dado que dicha clasificación no será posible, podemos pensar en utilizar propiedades sobre los objetos formales para definir lenguajes (en este caso, las gramáticas) para realizar una clasificación. En particular, la clasificación de Chomsky se construye a partir de la forma de las producciones de una gramática y establece una jerarquía (las gramáticas de un nivel están incluidos en los siguientes).

La jerarquía clásica de Chomsky consta de cuatro familias:

- **Tipo 0:** Gramáticas sin restricciones.

- **Tipo 1:** Gramáticas sensibles al contexto.
- **Tipo 2:** Gramáticas independientes del contexto.
- **Tipo 3:** Gramáticas regulares.

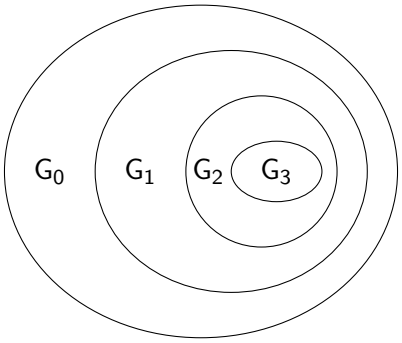


Figura 20: Jerarquía de Chomsky

En la Figura 20 podemos ver gráficamente la jerarquía de Chomsky. Podemos ver que las diferentes clases, están incluidas en las de nivel superior. Formalmente las clases son:

- **Gramáticas de Tipo 0:** también denominadas *sin restricciones*. Las producciones tienen la forma
$$\alpha \to \beta, \text{ donde } \alpha \in (V \cup T)^+, \ \beta \in (V \cup T)^*.$$
Notar que para estas producciones no se establece ningún tipo de restricción respecto a su forma.
- **Gramáticas de Tipo 1:** también denominadas *sensibles al contextp*. Las reglas de producción de estas gramáticas tienen la forma
$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2, \text{ donde } \alpha_1, \alpha_2 \in (V \cup T)^*, \ A \in V, \ \beta \in (V \cup T)^+.$$
En otras palabras, sólo se permite la sustitución del símbolo no terminal A por la cadena β cuando el símbolo A aparezca en el contexto indicado por α_1 y α_2.
- **Gramáticas de Tipo 2:** también denominadas gramáticas *libres de contexto* se caracterizan porque su conjunto de producciones se ajustan al siguiente esquema
$$A \to \alpha, \text{ donde } A \in V, \ \alpha \in (V \cup T)^*.$$
A diferencia de las gramáticas de tipo 1 resulta que el símbolo A siempre se puede sustituir por la cadena α independientemente del contexto en el que aparezca el símbolo A.
- **Gramáticas de Tipo 3:** o gramáticas *regulares*, son aquellas cuyas producciones tienen la forma
$$A \to xB, \text{ donde } A, B \in V, \ x \in T^*. \text{ Además, } B \text{ puede o no aparecer.}$$
Hay dos tipos de gramáticas regulares:

– *Lineales a la derecha*, con producciones de la forma

$$\begin{array}{lcl} A & \to & xB \\ A & \to & x \\ & & \text{donde } A,B \in V, x \in T^*. \end{array}$$

– *Lineales a la izquierda*, con producciones de la forma

$$\begin{array}{lcl} A & \to & Bx \\ A & \to & x \\ & & \text{donde } A,B \in V, x \in T^*. \end{array}$$

Como dijimos anteriormente cada gramática define un lenguaje, y a éstos también podemos clasificarlos (Figura 21).

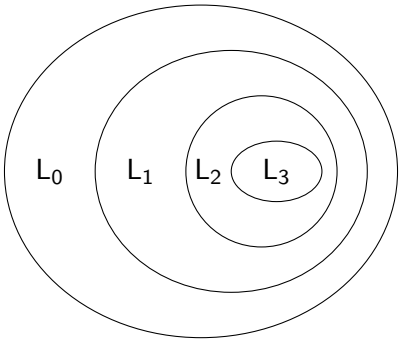


Figura 21: Jerarquía de Lenguajes

Decimos que un lenguaje pertenece a L_n si existe una gramática de tipo G_n capaz de generarlo, y dicha gramática es la más restrigida. Además de la definición de la jerarquía de Chomsky se deriva que:

$$\mathsf{L}_3 \subset \mathsf{L}_2 \subset \mathsf{L}_1 \subset \mathsf{L}_0.$$

Las clasificaciones introducidas anteriormente pueden ser refinadas, pero escapan a los objetivos de este apunte. En la siguiente sección nos centraremos en completar el estudio de los lenguajes regulares, introduciendo las gramáticas regulares. Luego extenderemos nuestra perspectiva con el estudio de lenguajes libres de contexto, una familia de lenguajes muy interesante para las ciencias de la computación.

4.3 Gramáticas regulares

Los lenguajes definidos por autómatas finitos o, lo que es lo mismo, por expresiones regulares, también pueden ser vistos como los lenguajes que generan ciertas gramáticas, llamadas gramáticas regulares.

Definición 4.4. Sea G una gramática tal que toda producción es de la forma

$$A \to aB \quad \text{o bien} \quad A \to \varepsilon,$$

donde A, B son variables y a es terminal. Entonces diremos que G es una *gramática regular*.

El lenguaje generado por un una gramática regular será llamado *lenguaje regular*.

Observemos primero que podemos considerar permitidas, en las gramáticas regulares, las producciones del tipo $A \to a$ pues se pueden obtener por composición de las permitidas en la definición. Una observación importante es que cualquier cadena con símbolos terminales y no terminales que se obtiene por derivación a partir del símbolo inicial, tiene a lo sumo una variable y ésta se ubica siempre en el extremo derecho de la cadena (gramáticas recursivas a derecha), o siempre en el extremo izquierdo (gramáticas recursivas a izquierda). Esto garantiza que cuando se dice que se usan ciertas producciones para realizar una derivación, no haya ambigüedad en la forma en que hay que aplicarla.

Ejemplo 4.3. Encontremos una gramática que genere el lenguaje asociado a la expresión regular a^*b^*. Es claro que el lenguaje asociado a a^*b^* es $L = \{a^n b^m : 0 \leq n,m\}$. Sea G gramática con una variable S, símbolos terminales a y b y las siguientes producciones:

$$\begin{aligned} S &\to aS \mid aT \mid bT \mid \varepsilon, \\ T &\to bT \mid \varepsilon \end{aligned}$$

entonces el lenguaje que genera G es L.

Demostración. Primero veamos que $L \subset L(G)$: sea $a^n b^m$ en L, si $n,m > 0$ usando la producción $S \to aS$, $(n-1)$-veces tenemos $S \Rightarrow a^{n-1}S$, luego usando la producción $S \to aT$ obtenemos $S \Rightarrow a^n T$, después usamos $S \to bT$ m-veces y obtenemos $S \Rightarrow a^n b^m T$, finalmente usando la producción $T \to \varepsilon$ tenemos la derivación $S \Rightarrow a^n b^m$. En el caso que $n = 0$ y $m > 0$ se hace $S \Rightarrow bT \Rightarrow b^m T \Rightarrow b^m$. El caso $n > 0$ y $m = 0$ es similar. Cuando $n = m = 0$, hacemos $S \Rightarrow \varepsilon$ usando la producción $S \to \varepsilon$.

Veamos ahora que $L(G) \subset L$: observemos que si usamos una derivación en esta gramática que agrega un b, entonces no se puede agregar más un a, como (por la definición de gramáticas regulares) los agregados de símbolos terminales sólo se pueden hacer a la derecha, es claro que las a's estarán todas a la izquierda de cualquier b.

□

Ejemplo 4.4. Encontremos una gramática regular G, cuyo lenguaje sea el lenguaje asociado a la expresión regular $r = (0+1)^*00$. Primero observemos que el lenguaje asociado a r es el de todas las cadenas de 0's y 1's que terminan en 00. Una gramática que genera el lenguaje $L(r)$ es la siguiente: los símbolos terminales serán $0,1$, las variables serán S,T y las producciones serán

$$S \to 0S \mid 1S \mid 0T, \qquad T \to 0U, \qquad U \to \varepsilon.$$

Demostración. Veamos primero que $L(r) \subset L(G)$: sea $x = a_1 a_2 \cdots a_k 00$ una cadena en $L(r)$, es decir que los a_i son 0 o 1 (en forma arbitraria). Si a_1 es 0, hacemos la derivación $S \Rightarrow 0S$, si es 1 hacemos $S \Rightarrow 1S$, es decir que tenemos $S \Rightarrow a_1 S$, en forma análoga obtenemos la derivación

$$S \Rightarrow a_1 S \Rightarrow a_1 a_2 S \Rightarrow \cdots \Rightarrow a_1 a_2 \cdots a_k S.$$

Si a $a_1 a_2 \cdots a_k S$ le aplicamos sucesivamente las producciones $S \to 0T$ y $T \to 0$, obtenemos $S \Rightarrow a_1 a_2 \cdots a_k S \Rightarrow a_1 a_2 \cdots a_k 0T \Rightarrow a_1 a_2 \cdots a_k 00$.

Veamos ahora que $L(G) \subset L(r)$: es claro que si $S \Rightarrow x$ una derivación con x compuesta de símbolos terminales, entonces la última producción que se usó fue $T \to 0$, y entonces la penúltima fue $S \to 0T$. Como las variables siempre están en el extremo derecho, la derivación es $S \Rightarrow x'S \Rightarrow x'0T \Rightarrow x'00 = x$. Es decir $x \in L(r)$.

□

Los ejemplos anteriores se pueden generalizar con el siguiente teorema, cuyo enunciado es un tanto complicado, pero que puede entenderse bien en casos concretos.

Teorema 4.1. *Sea r una expresión regular sobre una alfabeto Σ, entonces existe H una gramática regular con símbolos terminales en Σ tal que $L(r) = L(H)$. Más explícitamente: sea Σ un alfabeto.*

(1) $L(\emptyset) = \emptyset$ es $L(G)$ donde G es una gramática sin producciones.

(2) $L(\varepsilon) = \{\varepsilon\}$es $L(G)$ donde G tiene una única producción $S \to \varepsilon$.

(3) Para cada a en Σ, $L(a) = \{a\}$ es $L(G)$ donde G tiene producciones $S \to Ua$ y $T \to \varepsilon$.

Supongamos que r y r' son expresiones regulares tales que $L(r) = L(G)$, $L(r') = L(G')$con $G = (V,\Sigma,P,S)$ y $G' = (V',\Sigma,P',S')$. Supongamos además, sin perdida de generalidad, que $V \cap V' = \emptyset$, es decir que no tienen variables comunes. Debido a la definición de gramática regular, el conjunto P es de la forma:

$$\{C_k \to c_k D_k\} \cup \{U_t \to \varepsilon\},$$

para $k = 1,\ldots,m$, $t = 1,\ldots,s$. Entonces:

(4) $L(r+r')$ es $L(H)$, donde H tiene como variables $V = V \cup V'$, la variable inicial igual a S y con producciones $P_H = P \cup P' \cup \{S \to \alpha : \text{ si } S' \to \alpha\}$.

(5) $L(rr') = L(H)$, donde H es la gramática regular con variables $V_H = V \cup V'$, variable inicial igual a S y con producciones $P_H = P^{(1)} \cup P'$, donde $P^{(1)}$ es igual a:

$$\{C_k \to c_k D_k\} \cup \{C_k \to c_k S' : \text{si } D_k \to \varepsilon \in P\}$$

para $k = 1,\ldots,m$.

(6) $L(r^*) = L(H)$, donde H es la gramática regular con variables $V_H = V$,variable inicial S y producciones:

$$P_H = P \cup \{S \to \varepsilon\} \cup \{C_k \to c_k S : \text{si } C_k \to \varepsilon \in P\}$$

para $k = 1,\ldots,m$.

Demostración. La demostración es bastante directa, pero a su vez es tediosa. La dejamos como ejercicio para el lector. □

Ejercicio 4.1. Repetir los Ejemplos 4.3 y 4.4 usando el método del Teorema 4.1.

Como ya mencionamos, la recíproca del Teorema 4.1 también es verdadera, es decir, dado un lenguaje generado por una gramática regular G , existe una expresión regular r que denota el mismo lenguaje. En realidad esto lo probaremos de manera indirecta: es sencillo construir a partir de G un autómata no determinístico M con el mismo lenguaje que M. Luego par obtener r usamos el teorema de Kleene.

Proposición 4.2. *Sea $G = (V,\Sigma,P,S)$ una gramática regular, entonces existe $M = (Q,\Sigma,\delta,q_0,F)$ un NFA (sin ε-mov) tal que $L(M) = L(G)$. Explícitamente, $Q = V$, $q_0 = S$, $B \in \delta(A,a)$ sii $A \to aB$ y $F = \{A \in V : A \to \varepsilon\}$.*

La demostración es muy sencilla y se deja a cargo del lector.

4.4 Pumping Lemma

FiXme Note: revisar y completar

En esta sección veremos que existen lenguajes que no son regulares. Demostraremos esto usando un resultado conocido como el Pumping Lemma o "Lema del Inflado".

Proposición 4.3 (Pumping Lemma)**.** *Sea L un lenguaje regular. Entonces existe un número $k > 0$ tal que para toda cadena $\alpha \in L$ con $\mid \alpha \mid \geq k$, existen cadenas β_1, β_2, γ con $\mid \beta_1\gamma \mid \leq k$, $\mid \gamma \mid > 0$ que satisfacen*

1. $\alpha = \beta_1\gamma\beta_2$,
2. $\beta_1\gamma^n\beta_2 \in L$ *para todo* $n \geq 1$.

Demostración. Debido a que L es un lenguaje regular, existe M un DFA tal que $L = L(M)$. Sea k el número de estados de M y α una cadena en L con $\mid \alpha \mid \geq k$. Entonces $\alpha = a_{i_1} \cdots a_{i_t}$ donde a_{i_j} símbolo de input ($1 \leq j \leq t$) y $t \geq k$. Sean $q_{i_0}, q_{i_1}, \ldots, q_{i_t}$ estados tal que, $q_{i_0} = q_0$ el estado inicial, q_{i_t} un estado final y

$$q_{i_0} \xrightarrow{a_{i_1}} q_{i_1} \xrightarrow{a_{i_2}} \cdots \xrightarrow{a_{i_{t-1}}} q_{i_{t-1}} \xrightarrow{a_{i_t}} q_{i_t}.$$

Es claro que, como $q_{i_0}, \ldots, q_{i_k}$ es una lista de $k+1$ estados y como el autómata tiene k estados, hay al menos dos estados que se repiten en la lista, digamos q_{i_r} y q_{i_s}, con $r < s \leq k$. Sean entonces $\beta_1 = a_{i_1} \cdots a_{i_{r-1}}$, $\gamma = a_{i_r} \cdots a_{i_{s-1}}$ y $\beta_2 = a_{i_s} \cdots a_{i_t}$. Luego, $\alpha = \beta_1\gamma\beta_2$, $\mid \gamma \mid > 0$ y $\mid \beta_1\gamma \mid \leq k$. Observemos que

$$q_{i_0} \xrightarrow{\beta_1} q_{i_r} \xrightarrow{\gamma} q_{i_r} \xrightarrow{\beta_2} q_{i_t},$$

en particular tenemos que $q_{i_r} \xrightarrow{\gamma} q_{i_r}$. Por lo tanto, se cumple que para $n > 0$,

$$q_{i_r} \xrightarrow{\gamma^n} q_{i_r} \quad \text{o equivalentemente} \quad q_{i_r} \xrightarrow{\gamma} q_{i_r} \xrightarrow{\gamma} \cdots \xrightarrow{\gamma} q_{i_r} \ (n\text{-veces}).$$

Por consiguiente, también vale

$$q_{i_0} \xrightarrow{\beta_1} q_{i_r} \xrightarrow{\gamma^n} q_{i_r} \xrightarrow{\beta_2} q_{i_t},$$

lo cual implica que $\beta_1\gamma^n\beta_2$ es una cadena aceptada por M y por lo tanto que pertenece a L. Esto prueba el Pumping Lemma. □

Como mencionamos arriba el Pumping Lemma es útil para demostrar que un lenguaje no es regular.

Ejemplo 4.5. Probemos que el lenguaje $L = \{a^n b^n : n > 1\}$ no es regular.

Demostración. Hagamos la prueba por el absurdo suponiendo que L es regular. Sea k como en el Pumping Lemma. Debemos elegir $\alpha \in L$ de tal forma de obtener una contradicción que nos lleve al absurdo. Elegimos $\alpha = a^k b^k$, por el Pumping Lemma existen cadenas β_1, β_2, γ con $\mid \beta_1\gamma \mid \leq k$, $\mid \gamma \mid > 0$ que satisfacen $\alpha = \beta_1\gamma\beta_2$ y $\beta_1\gamma^n\beta_2 \in L$ para todo $n \geq 1$. Como $\mid \beta_1\gamma \mid \leq k$, γ es de la forma a^s con $1 \leq s$, luego $\alpha = a^r a^s a^t b^k$ con $r + s + t = k$ y $s > 0$. Además $a^r a^{ns} a^t b^k \in L$ para $n \geq 1$. Esto es una contradicción pues como $r + ns + t > k$ para $n > 1$, entonces $a^r a^{ns} a^t b^k \notin L$ para $n > 1$. □

El Pumping Lemma es utilizado para detectar si un lenguaje no es regular, con la siguiente estrategia:

1. Seleccionar el lenguaje L que usted desea ver que no es regular.

2. Suponer que es regular y que por lo tanto existe un k como en el Pumping Lemma.

3. Seleccionar una cadena α en L de longitud mayor que k. La elección de la cadena no es arbitraria y depende del lenguaje dado.

4. Descomponer la cadena de acuerdo al Pumping Lemma y generar nuevas cadenas en L haciendo "inflación" en el centro (es decir las cadenas $\beta_1\gamma^n\beta_2$).

5. Verificar que las cadenas "infladas" no pertenecen a L. Esto genera una contradicción que vino de suponer que L era regular.

Observar que la estrategia anterior podría no servir, pero el hecho de que no podamos aplicar el Pumping Lemma a un lenguaje L no implica que este sea regular.

Ejemplo 4.6. Demostrar que el lenguaje $L = \{0^n001^n \mid n \in \mathbb{N}\}$ no es un lenguaje regular.

Demostración. Supongamos que L es un lenguaje regular y sea k como en el Pumping Lemma. La cadena $\alpha = 0^k001^k$ es de L y por lo tanto es aceptada por el autómata. Por el Pumping Lemma $\alpha = \beta_1\gamma\beta_2$, con $\mid \gamma \mid > 0$ y $\mid \beta_1\gamma \mid \leq k$ y tal que $\beta_1\gamma^n\beta_2$ es aceptado por el autómata para todo $n > 0$. Como $\mid \beta_1\gamma \mid \leq k$, es claro que $\beta_1 = 0^r$, $\gamma = 0^s$ y $s \geq 1$. Es decir $\beta_1\gamma^n\beta_2 = 0^{k+1+(n-1)s}001^{k+1}$. Por lo tanto, $0^{k+1}0^{(n-1)s}001^{k+1}$ es una cadena de L para todo $n > 0$. Lo cual es un absurdo, pues esa cadena no pertenece a L para $n > 1$. □

Observar que en el ejemplo anterior hay cadenas de L que son de longitud mayor que k (por ejemplo $0^{k/2}001^{k/2}$, si k es par) que no sirven para demostrar, usando el Pumping Lemma, que el lenguaje no es regular. Es decir, se debe tener mucho cuidado en la elección de la cadena, pues la elección de una cadena incorrecta hará que la utilización del Pumping Lemma no nos sirva para demostrar que el lenguaje no es regular.

4.5 Gramáticas Libres de Contexto

Una *gramática libre de contexto* es un conjunto finito de *variables* (también llamadas *no terminales* o *categorías sintácticas*), un conjunto de símbolos llamados *terminales* y un conjunto de reglas llamadas *producciones* que transforman una variable en una cadena formada por variables y terminales.

La motivación original de las gramáticas libres de contexto fue la descripción de los lenguajes naturales. Recordemos el ejemplo de la sección anterior:

$$\begin{aligned}
< \text{oración} > &\rightarrow < \text{sujeto} >< \text{predicado} > \\
< \text{sujeto} > &\rightarrow < \text{sujeto} >< \text{adjetivo} > \\
< \text{sujeto} > &\rightarrow \text{el perro} \\
< \text{adjetivo} > &\rightarrow \text{pequeño} \\
< \text{adjetivo} > &\rightarrow \text{grande} \\
< \text{adjetivo} > &\rightarrow \text{bueno} \\
< \text{predicado} > &\rightarrow < \text{verbo} >< \text{adjetivo} > \\
< \text{verbo} > &\rightarrow \text{es}
\end{aligned}$$

donde las categorías sintácticas son denotadas por los $<, >$ y los terminales son "el perro", "pequeño", "grande", "bueno" y "es".

Pese a su origen, las gramáticas libres de contexto no son adecuadas para la descripción de lenguajes naturales, una razón importante de este hecho es que es necesaria información semántica, además

de la sintáctica, para poder construir frases correctas en castellano. Por ejemplo, aunque las reglas de producción anteriores son aparentemente "naturales" también podemos armar la siguiente frase "el perro grande es pequeño".

Ahora, formalizaremos los conceptos expresados previamente.

Definición 4.5. Una *gramática libre de contexto (CFG)* es una 4-upla $G = (V, T, P, S)$ tal que cada producción en P es de la forma $A \to \alpha$, donde A es una variable y α es una cadena formada por variables y terminales (que puede ser vacía), es decir $\alpha \in (V \cup T)^*$.

El lenguaje generado por una CFG será llamando *lenguaje libre de contexto*.

Ejemplo 4.7. Consideremos la gramática con una variable E, símbolos terminales $+$, $*$, $($, $)$ y id, y producciones

$$E \to E + E \mid E * E \mid (E) \mid id \quad .$$

Entonces $(id + id) * id$ se deriva de E, pues

$$\begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow (E) * E \\ &\Rightarrow (E) * id \\ &\Rightarrow (E + E) * id \\ &\Rightarrow (E + id) * id \\ &\Rightarrow (id + id) * id \end{aligned}$$

La primera línea se obtiene usando la producción $E \to E * E$, la segunda se obtiene reemplazando la primera E por el lado derecho de la producción $E \to (E)$. Las restantes líneas se obtienen de aplicar sucesivamente las producciones $E \to id$, $E \to E + E$, $E \to id$, y $E \to id$.

Ejemplo 4.8. Consideremos la gramática con una variable S , símbolos terminales a y b, y producciones $S \to aSb \mid ab$. Veamos que el lenguaje generado por esta gramática es el de todas las cadenas en el alfabeto $\{a, b\}$ que son de la forma $a^n b^n$ con $n > 0$.

Demostración. Hay esencialmente una única forma de obtener una cadena con símbolos terminales: primero aplicar repetidas veces la producción $S \to aSb$ y luego terminar con la producción $S \to ab$. Es decir que las derivaciones son del tipo

$$S \Rightarrow aSb \Rightarrow a^2Sb^2 \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n.$$

Entonces es claro que las palabras generadas por el lenguaje son de la forma $a^n b^n$ y obviamente toda cadena de la forma $a^n b^n$ se obtiene haciendo la derivación escrita más arriba. □

Ejemplo 4.9. Consideremos la gramática con una variable S, símbolos terminales a y b, y producciones $S \to aSa \mid bSb \mid a \mid b \mid \varepsilon$. Veamos que el lenguaje generado por esta gramática es el de todas las cadenas en el alfabeto $\{a, b\}$ que son capicúa.

Demostración. Por definición, una palabra capicúa es o bien de la forma $x_1x_2 \ldots x_n x_n \ldots x_2x_1$ o de la forma $x_1x_2 \ldots x_{n-1}x_n x_{n-1} \ldots x_2x_1$, con x_i igual a a o b. Usando sucesivamente las producciones $S \to x_1Sx_1$, $S \to x_2Sx_2$, …, $S \to x_{n-1}Sx_{n-1}$, obtenemos la derivación

$$S \Rightarrow x_1Sx_1 \Rightarrow x_1x_2Sx_2x_1 \Rightarrow \cdots \Rightarrow x_1x_2 \ldots x_{n-1}Sx_{n-1} \ldots x_2x_1,$$

si ahora usamos las producción $S \to x_n S x_n$ y luego $S \to \varepsilon$ obtenemos $S \Rightarrow x_1 x_2 \ldots x_n x_n \ldots x_2 x_1$. Por otro lado si aplicamos la producción $S \to x_n$ a $x_1 x_2 \ldots x_{n-1} S x_{n-1} \ldots x_2 x_1$ obtenemos la derivación $S \Rightarrow x_1 x_2 \ldots x_{n-1} S x_{n-1} \ldots x_2 x_1 \Rightarrow x_1 x_2 \ldots x_{n-1} x_n x_{n-1} \ldots x_2 x_1$.

Veamos por inducción la recíproca, es decir que una cadena del lenguaje generado por la gramática es capicúa: la hipótesis inductiva es $S \Rightarrow \alpha S \beta$, entonces $\alpha = x_1 x_2 \ldots x_k$ y $\beta = x_k \ldots x_2 x_1$, con x_i igual a a o b. La inducción se hace sobre la longitud de α. Si la longitud de α es 1 el resultado es trivial. Supongamos que tenemos probado el resultado para cadenas de longitud $k-1$. Sea $S \Rightarrow \alpha S \beta$, con $|\alpha| = k$. Sea $\alpha = x_1 x_2 \ldots x_k$, es claro que la última producción que se usó es $S \to x_k S x_k$, luego tenemos $S \Rightarrow \alpha' S \beta' \Rightarrow \alpha' x_k S x_k \beta' = \alpha S \beta$. Claramente $\alpha' = x_1 x_2 \ldots x_{k-1}$ tiene longitud $k-1$, y entonces por hipótesis inductiva tenemos que $\beta' = x_{k-1} \ldots x_2 x_1$ y por consiguiente $\beta = x_k \ldots x_2 x_1$. Sea ahora una cadena α de símbolos terminales, tal que $S \Rightarrow \alpha$, entonces la derivación es $S \Rightarrow \alpha_1 S \alpha_2 \Rightarrow \alpha_1 x \alpha_2 = \alpha$, con x igual a a, b o ε. Por lo visto más arriba $\alpha_1 x \alpha_2 = \alpha$ es capicúa. □

Una manera alternativa de enunciar las producciones de una gramática es por el empleo de la *forma normal de Backus-Naur* o *BNF*. En una BNF los símbolos no terminales empiezan con "<" y terminan con ">". Las producción $A \to \alpha$ se expresa $A ::= \alpha$.

Ejemplo 4.10. Un entero es una cadena que consiste de un símbolo opcional ($+$ o bien $-$) seguido por un entero o cadena de dígitos (del 0 al 9). La siguiente gramática (escrita en notación BNF) genera todos los enteros.

$$\begin{aligned}
< \text{dígito} > &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
< \text{entero} > &::= < \text{entero con signo} > \mid < \text{entero sin signo} > \\
< \text{entero con signo} > &::= + < \text{entero sin signo} > \mid - < \text{entero sin signo} > \\
< \text{entero sin signo} > &::= < \text{dígito} > \mid < \text{dígito} > < \text{entero sin signo} >
\end{aligned}$$

El símbolo inicial es $<$ entero $>$, las otras variables son $<$ entero con signo $>$, $<$ entero sin signo $>$, $<$ dígito $>$, los símbolos terminales son $0,1,2,3,4,5,6,7,8,9,+,-$.

Por ejemplo la derivación del entero -452 es

$$\begin{aligned}
< \text{entero} > &\Rightarrow < \text{entero con signo} > \\
&\Rightarrow - < \text{entero sin signo} > \\
&\Rightarrow - < \text{dígito} > < \text{entero sin signo} > \\
&\Rightarrow - < \text{dígito} > < \text{dígito} > < \text{entero sin signo} > \\
&\Rightarrow - < \text{dígito} > < \text{dígito} > < \text{dígito} > \\
&\Rightarrow -4 < \text{dígito} > < \text{dígito} > \\
&\Rightarrow -45 < \text{dígito} > \\
&\Rightarrow -452
\end{aligned}$$

Claramente el lenguaje asociado a esta gramática es el de todas las cadenas que empiezan con $+$ o $-$ y sigue una sucesión de dígitos o las cadenas que son una sucesión de dígitos.

La sintaxis de algunos lenguajes de computación de alto nivel, como Pascal, C o FORTRAN, pueden expresarse en BNF.

4.6 Formas normales de Chomsky y Greibach

Dada una gramática libre de contexto G es posible hallar gramáticas con producciones de cierto tipo tal que el lenguaje generado por estas gramáticas sea del mismo que el generado por G.

Definición 4.6. Sea $G = (V,T,P,S)$ una CFG. Diremos que G está en la *forma normal de Chomsky* si todas las producciones son de la forma

$$A \to a \qquad \text{o} \qquad A \to BC,$$

donde $A,B,C \in V$ y $a \in T$. Es decir, si el lado derecho de todas las producciones es o bien un símbolo terminal o dos variables. Diremos que G está en la *forma normal de Greibach* si todas las producciones son de la forma

$$A \to a\alpha,$$

donde $A \in V$, $a \in T$ y $\alpha \in V^*$. Es decir, si el lado derecho de todas las producciones es un símbolo terminal seguido de ninguna o varias variables.

Dada una gramática G es posible hallar gramáticas G' y G'' en forma normal de Chomsky y Greibach, respetivamente, tal que el lenguaje generado por G sea el mismo que el generado por G' o G''.

En lo que resta de la sección veremos como dada una gramática G libre de contexto podremos encontrar en forma algorítmica una gramática equivalente en la forma de Chomsky. El procedimiento consta de los siguientes pasos:

1. Eliminar ε-producciones ($A \to \varepsilon$).
2. Eliminar variables que no llevan a cadenas de símbolos terminales.
3. Eliminar símbolos (variables y terminales) que no pueden ser alcanzados.
4. Eliminar producciones unitarias ($A \to B$).
5. Forma normal de Chomsky.

Comencemos a hacer el procedimiento:

(1) Eliminar ε-producciones. Con este procedimiento obtendremos una gramática con lenguaje $L(G) - \{\varepsilon\}$. Si el lenguaje original contenía a ε, entonces al final agregaremos la producción $S \to \varepsilon$. El procedimiento es como sigue. Primero sea $N = \emptyset$ y ahora:

(a) si $A \to \varepsilon$ es producción, entonces agregar A a N.

(b) Iteramos el siguiente paso hasta que N se estabilice: si $A \to \alpha$ es producción con $\alpha \in N^+$, entonces agregar A a N.

Es claro que el proceso (b) termina debido a que $N \subset V$.

El nuevo conjunto P' de producciones es el siguiente: si $A \to X_1X_2\ldots X_k$ es una producción en P donde $X_i \in V \cup T$, entonces $A \to \alpha_1\alpha_2\ldots\alpha_k$ está en P' si

1. $\alpha_i = X_i$ si $X_i \notin N$,
2. $\alpha_i = X_i$ o $\alpha_i = \varepsilon$ si $X_i \in N$,
3. $\alpha_1\alpha_2\ldots\alpha_k \neq \varepsilon$.

Es decir, si partimos de $P' = \emptyset$, para toda $A \to \alpha$ en P, se agregan a P' las producciones $A \to \alpha'$ donde $\alpha' \neq \varepsilon$ y α' es cualquier combinación que resulte de eliminar algunas variables en N de α . Observar que si S está en N, entonces ε es aceptado por el lenguaje original.

(2) Eliminar variables que no llevan a cadenas de símbolos terminales. Debemos reemplazar el conjunto de variables originales por el conjunto V' que se construye con cierto número de iteraciones: el primer V' se obtiene de agregar todas las variables que figuren del lado izquierdo de producciones del tipo $A \to \beta$ donde $\beta \in T^+$. Luego se itera el siguiente procedimiento hasta que se estabilice V'

(a) el nuevo V' se obtiene agregando todas las variables que figuren del lado izquierdo de producciones del tipo $A \to \beta$ donde $\beta \in$, es decir cuando β es una cadena donde las variables que aparecen son de V'. En pseudocódigo sería

$$V' := V' \cup \{A : A \to \beta, \beta \in (T \cup V')^+\}$$

Terminada la iteración nuestro nuevo conjunto de variables V será igual al último V'. Si S no está en V', entonces es fácil ver que el lenguaje generado por G es vacío.

(3) Eliminar símbolos que no pueden ser alcanzados. Sea $V' = \{S\}$ y $T' = \emptyset$. Iterar de la siguiente forma hasta que V' y T' se estabilicen:

(a) si $A \to \alpha$ es una producción con $A \in V'$, entonces agregar a V' todas las variables que se encuentren en α y agregar a T' todos los símbolos no terminales que se encuentren en α. En pseudocódigo sería

$$V' := V' \cup \{B \in V : A \to \alpha B \beta,\ A \in V',\ \alpha, \beta \in (T \cup V)^*\}$$
$$T' := T' \cup \{a \in T : A \to \alpha a \beta,\ A \in V',\ \alpha, \beta \in (T \cup V)^*\}$$

La iteración finaliza puesto que $V' \subset V$ y $T' \subset T$. Nuestros nuevos V y T serán V' y T', respectivamente, obtenidos por las iteraciones de más arriba.

(4) Eliminar producciones unitarias. Las producciones unitarias son aquellas de la forma $A \to B$ con $A, B \in V$. El proceso de eliminar producciones unitarias es sencillo: recorremos el conjunto de producciones y cada vez que encontramos una producción del tipo $A \to B$ la eliminamos y todas las producciones $B \to \gamma$ las cambiamos por $A \to \gamma$.

(5) Forma normal de Chomsky. En este caso suponemos que hemos completado los procedimientos (1)…(4). Si $A \to \alpha$ es una producción la *longitud de* $A \to \alpha$ se define como la longitud de la cadena α. Para hacer la forma normal de Chomsky procederemos de la siguiente manera:

(a) por cada símbolo terminal a se agrega una nueva variable A y una producción $A \to a$. Ahora, en cada producción de longitud mayor que 1 se reemplaza cada símbolo terminal a por la nueva variable A. Después de hacer esto todas las producciones son del tipo $A \to a$, con a terminal o del tipo $A \to A_1 \ldots A_n$ con $A_1, \ldots, A_n$ variables.

(b) En cada producción de longitud mayor que dos $A \to A_1 A_2 \ldots A_n$ se reemplaza $A_1 A_2$ por una nueva variable B y se agrega la producción $B \to A_1 A_2$. Este procedimiento se itera hasta que no haya producciones de longitud mayor que 2. Es claro que la iteración termina debido as que en cada paso se disminuye la longitud máxima que pueden tener las producciones.

Ejemplo 4.11. Sea $G = (V, T, P, S)$ con $V = \{S, A\}$, $T = \{a, b\}$ y producciones

$$S \to aAS \mid a \qquad A \to SbA \mid SS \mid ba$$

Ejemplo 4.12. Sea $G = (V,T,P,S)$ con $V = \{S,A\}$, $T = \{a,b\}$ y producciones

$$S \to aAS \mid a \qquad A \to SbA \mid SS \mid ba$$

Encontrar una gramática en la forma normal de Chomsky que genere el lenguaje $L(G)$.

No es difícil verificar que esta gramática no necesita que realicemos los procedimientos (1), (2), (3) y (4). Debemos entonces aplicar el procedimiento (5):

(a) Primero agregamos una variable por cada símbolo terminal, X por a e Y por b. También agregamos las producciones $X \to a$ e $Y \to b$. En las producciones de longitud mayor que 1 reemplazamos a por X y b por Y. Obtenemos entonces la gramática $G = (V,T,P,S)$ con $V = \{S,A,X,Y\}$, $T = \{a,b\}$ y producciones

$$\begin{array}{llll} S & \to XAS \mid a & A & \to SYX \mid SS \mid YX \\ X & \to a & T & \to b \end{array}$$

(b) Reemplazamos en $S \to XAS$ a XA por U (una nueva variable) y agregamos la producción $U \to XA$. Reemplazamos en $A \to SYX$ a SY por W (una nueva variable) y agregamos la producción $W \to SY$. Obtenemos la gramática $G = (V,T,P,S)$ con $V = \{S,A,X,Y,U,W\}$, $T = \{a,b\}$ y producciones

$$\begin{array}{llll} S & \to US \mid a & A & \to WX \mid SS \mid YX \\ X & \to a & T & \to b \\ U & \to XA & W & \to SY \end{array}$$

La formal normal de Greibach se puede obtener a partir de la forma normal de Chomsky, pero es mucho más complicado hacerlo y queda fuera de los alcances de este escrito.

5 Autómatas con pila

Hemos visto que los lenguajes que generan las expresiones regulares o, equivalentemente, los lenguajes regulares se pueden obtener a partir de autómatas finitos, y viceversa. En forma análoga, a los lenguajes libres de contexto le corresponden los autómatas con pila. En esta sección veremos la definición de los autómatas con pila, los lenguajes generados por ellos y algunos ejemplos. No probaremos, por estar fuera de los alcances de este texto, la equivalencia entre los lenguajes libres de contexto y los lenguajes aceptados por los autómatas con pila.

Un autómata con pila (PDA) es esencialmente un autómata finito que posee control sobre una pila, es decir una lista de la cual solo se puede "leer", "poner" o "sacar" el primer elemento. Dado el estado actual del autómata y el primer elemento de la pila, un símbolo de input nos llevará (posiblemente en forma no determinística) el estado siguiente y a la modificación que se debe hacer en el primer elemento de la pila. Diremos que una cadena es aceptada por *pila vacía* por el PDA si cuando la aplicamos obtenemos una pila vacía. Diremos que una cadena es aceptada por *estado final* por el PDA si lleva el estado inicial a uno final. Los lenguajes aceptados por los autómatas con pila, tantos los aceptados por pila vacía o por estado final, son los mismos que los aceptados por las gramáticas libres de contexto e incluyen estrictamente a los lenguajes regulares.

Ejemplo 5.1. El lenguaje $L = \{wcw^R : w \in (0+1)^*$ y c símbolo $\}$ es un lenguaje no regular. Esto es fácil de ver usando el Pumping Lemma. Sin embargo, es un lenguaje libre de contexto generado por la gramática $S \to 0S0 \mid 1S1 \mid c$. Mostraremos a continuación un autómata con pila cuyo lenguaje por

pila vacía es L. Los símbolos de input serán, obviamente, 0, 1 y c. Consideremos dos estados q_1 y q_2 y que en la pila se pueden "apilar" tres tipos de objetos rojos (R), verdes (V) y azules (A). La idea para definir el autómata es la siguiente: la pila comenzará con un solo elemento R para indicar, justamente, el comienzo de la pila. Ahora, pensemos a la pila como una "memoria" donde guardaremos la forma de la primera porción de la palabra hasta c: haremos que cada vez que el input sea 0, se agregue a la pila una A y cada vez que sea 1 se agregue una V. Estos inputs no cambian el estado inicial. Cuando ingresa el input c, cambiamos de estado para indicar que tenemos que empezar a leer el final de la palabra. Ahora deberemos desapilar convenientemente, de tal forma que si después de c viene w^R, la pila quede vacía. Por ejemplo, supongamos que w termina en 1, entonces, después de aplicar c, estamos en el segundo estado y la pila tiene en la parte superior una V. Esto indica que la última letra de w es un 1 y por lo tanto, para tener esperanza de que la palabra sea aceptada, la primera letra después de c debería ser un 1. Por lo tanto decimos que si el input es 1 y la pila muestra V, se retira V. Análogamente si en la parte superior hay un A y el input es 0, se retira A. Siguiendo así, si el sufijo de c es w^R, llegaremos a una pila con un solo elemento, el R. El último movimiento, un ε-movimiento, se define de la siguiente manera: si estamos en el segundo estado y la pila muestra el R, se saca el R. Resumiendo: el autómata tendrá las siguientes reglas:

1. Comenzamos en el estado q_1 y con R en la pila.

2. En el caso en que el estado es q_1 el autómata actúa de la siguiente manera. Si se ingresa el símbolo de input 0, agregamos a la pila una A. Si el símbolo de input es 1 agregamos una V. En ambos caso el estado permanece q_1. Si la entrada es c pasamos al estado q_2 y la pila no cambia.

3. En el caso en que el estado es q_2 el autómata actúa de la siguiente manera. Si se ingresa el símbolo de input 0 y el primero de la pila es A, se retira el primer elemento de la pila (es decir la A). Si el símbolo de input es 1 y el primero de la pila es V, se retira el primer elemento de la pila. Si el primer elemento de la pila es R, se lo retira sin esperar input. En todos los casos el estado continúa siendo q_2.

4. En las situaciones no contempladas en los items anteriores, el autómata no hace nada.

Haciendo algunos ejemplos con cadenas particulares es fácil convencerse que las únicas cadenas aceptadas por pila vacía son las de la forma wcw^R.

Definición 5.1. Un *autómata con pila* es una 7-upla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, en donde

1. Q es un conjunto finito de *estados*;

2. Σ es un alfabeto, el *alfabeto de entrada*;

3. Γ es un alfabeto, el *alfabeto de la pila*;

4. $q_0 \in Q$, el *estado inicial*;

5. $Z_0 \in \Gamma$, el *símbolo inicial* de la pila;

6. $F \subset Q$, los *estados finales*;

7. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_f(Q \times \Gamma^*)$, la *función de transición*, donde $\mathcal{P}_f(Q \times \Gamma^*)$ indica los subconjuntos finitos de $Q \times \Gamma^*$.

Usaremos PDA (por sus siglas en inglés) como sinónimo de "autómata con pila".

En general haremos uso de letras minúsculas del comienzo del alfabeto $(a,b,c,\ldots)$ para denotar los símbolos del alfabeto de entrada (es decir de Σ) y usaremos letras minúsculas, pero del final del alfabeto $(\ldots,x,y,z)$, para denotar cadenas en Σ. Las letras mayúsculas $(A,B,C,\ldots,X,Y,Z)$, denotarán símbolos de la pila (es decir elementos de Γ) y las letras griegas minúsculas $(\alpha,\beta,\gamma,\ldots)$ denotarán elementos de Γ^*.

Observación 5.1. Debemos hacer algunos comentarios acerca de como se debe interpretar δ. Si tenemos, por ejemplo, que $\delta(q,a,Z)=\{(p,\gamma)\}$ esto lo debemos interpretar de la siguiente manera: si q estado y Z cima de la pila, al aplicarle a del alfabeto de entrada, el estado del autómata cambia a p y en la pila se produce el reemplazo de Z por γ, quedando el primer símbolo de γ como cima de la pila. Por ejemplo si estamos en un estado q y la pila es $Z_1Z_2Z_2$, entonces aplicar a usando la regla $\delta(q,a,Z_1)=\{(p,Z_2Z_1Z_2)\}$, hace que pasemos al estado p y que la pila pase a ser $Z_2Z_1Z_2Z_2Z_2$.

Los autómatas con pila permiten acciones no determinísticas, pues

$$\delta(q,a,Z)=\{(p_1,\gamma_1),\ldots,(p_m,\gamma_m)\}$$

indica la posibilidad de hacer diferentes acciones aún con el mismo input. Por otro lado, también están permitidos los ε-movimientos, es decir

$$\delta(q,\varepsilon,Z)=\{(p_1,\gamma_1),\ldots,(p_m,\gamma_m)\}$$

está permitido y permite, sin ningún input, cambiar el estado y la pila.

Notemos que los DFA y NFA definidos en capítulos anteriores son casos especiales de autómatas con pila: basta "olvidarse" de la pila en la definición.

Ejemplo 5.2. Describamos en lenguaje formal el autómata con pila del Ejemplo 5.1: $M=(\{q_1,q_2\},\{0,1,c\},\{A,V,R\},\delta,q_1,R,\emptyset)$. Observemos que hemos determinado que el conjunto de estados finales es vacío. Esto se debe a que en este caso estamos interesados en el lenguaje aceptado por pila vacía y por lo tanto los estados finales son irrelevantes. La descripción de δ es

$$\begin{array}{lll}\delta(q_1,0,X)=\{(q_1,AX)\}, & \delta(q_1,1,X)=\{(q_1,VX)\}, & \delta(q_1,c,X)=\{(q_2,X)\},\\ \delta(q_2,0,A)=\{(q_2,\varepsilon)\}, & \delta(q_2,1,V)=\{(q_2,\varepsilon)\}, & \delta(q_2,\varepsilon,R)=\{(q_2,\varepsilon)\},\end{array}$$

donde X es un símbolo arbitrario de la pila. Las transiciones no descritas son $\delta(q,a,Z)=\emptyset$.

Dada una cadena del alfabeto de entrada queremos describir formalmente la situación del autómata con pila después de haberse aplicado parte de la cadena. Para ello debe registrarse, sin duda, el estado actual del autómata y el contenido de la pila. Además, registraremos la parte de la cadena que todavía no ha sido aplicada. Formalmente:

Definición 5.2. Sea $M=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ un PDA. Una *descripción instantánea (ID)* es un triple (q,w,γ), donde $q\in Q$, $w\in\Sigma^*$ y $\gamma\in\Gamma^*$.

Si $(q,aw,Z\gamma)$ y $(p,w,\beta\gamma)$ son dos ID, denotaremos

$$(q,aw,Z\gamma)\vdash(p,w,\beta\gamma)$$

si $(p,\beta)\in\delta(q,a,Z)$. Dadas (q,w,γ) y (q',w',γ') descripciones instantáneas, denotaremos

$$(q,w,\gamma)\vdash^*(q',w',\gamma')$$

si existen $(q_1,w_1,\gamma_1),\ldots,(q_m,w_m,\gamma_m)$ descripciones instantáneas tales que

$$(q,w,\gamma)\vdash(q_1,w_1,\gamma_1)\vdash\cdots\vdash(q_m,w_m,\gamma_m)\vdash(q',w',\gamma').$$

Por convención, será aceptado $(q,w,\gamma)\vdash^*(q,w,\gamma)$.

Las descripciones instantáneas serán útiles para definir el lenguaje aceptado por un PDA.

Definición 5.3. Sea $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Entonces, definimos $L(M)$ el *lenguaje de M por estado final*, como

$$L(M) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (p, \varepsilon, \gamma) \text{ para algún } p \in F, \gamma \in \Gamma^*\}.$$

El *lenguaje de M por pila vacía* es:

$$N(M) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon) \text{ para algún } p \in Q\}.$$

Como ya hemos mencionado el conjunto de los lenguajes aceptados por estado final es equivalente al conjunto de los lenguajes aceptados por pila vacía. La demostración de este hecho sigue el espíritu de las demostraciones que hemos visto en teoría de lenguajes, dado un lenguaje definido de cierta manera, construimos en forma algorítmica un autómata que acepta ese lenguaje.

Recordemos que si M es un NFA, puede asociarse trivialmente M' un PDA, que esencialmente es el mismo M con una pila que no se usa. Es claro entonces, por las respectivas definiciones, que el lenguaje de M coincide con el lenguaje de M' por estado final.

Ejemplo 5.3. $L = \{aa^R : a \in \{0,1\}^*\}$

En la sección 4.6 hemos visto que toda gramática se puede reducir a una gramática equivalente en la forma normal de Greibach. La siguiente proposición probará el "implica" de la equivalencia entre lenguajes generados por gramáticas libres de contexto y lenguajes aceptados por autómatas con pila.

Proposición 5.1. *Sea G una CFG en la forma normal de Greibach. Entonces existe M un PDA tal que* $L(G) = N(M)$.

Demostración. Denotemos $L = L(G)$ y hagamos la demostración en el caso en que ε no esté en L. La demostración en el otro caso es similar y se deja a cargo del lector. G está en la forma normal de Greibach, es decir toda producción es de la forma $A \to a\gamma$ con $\gamma \in V^*$. Definamos el PDA

$$M = (\{q\}, T, V, \delta, q, S, \{\emptyset\}),$$

donde $(q, \gamma) \in \delta(q, a, A)$ si y sólo si $A \to a\gamma$ está en P. Observemos que en este caso los estados del autómata no tiene importancia (siempre estamos en el mismo estado) y lo único que importa es la pila. Las operaciones elementales que hace δ sobre la pila claramente simulan las producciones y una composición de estas operaciones simula una derivación a izquierda. De manera formal debemos demostrar que

$$S \Rightarrow w \qquad \text{si y sólo si} \qquad (q, w, S) \vdash^* (q, \varepsilon, \varepsilon),$$

para $w \in T^*$. En realidad es más sencillo demostrar

$$S \Rightarrow w\gamma \qquad \text{si y sólo si} \qquad (q, w, S) \vdash^* (q, \varepsilon, \gamma), \tag{4}$$

para $w \in T^*$ y $\gamma \in V^*$ y se deja como ejercicio para el lector. Claramente, esto implica lo anterior haciendo $\gamma = \varepsilon$. □

Finalizaremos la sección dando la definición de autómatas con pila determinísticos.

Definición 5.4. Un *autómata con pila determinístico* es una 7-upla $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, en donde

1. Q es un conjunto finito de *estados*;
2. Σ es un alfabeto, el *alfabeto de entrada*;
3. Γ es un alfabeto, el *alfabeto de la pila*;
4. $q_0 \in Q$, el *estado inicial*;
5. $Z_0 \in \Gamma$, el *símbolo inicial* de la pila;
6. $F \subset Q$, los *estados finales*;
7. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \{\emptyset\} \cup (Q \times \Gamma^*)$, la *función de transición*, tal que si $\delta(q, \varepsilon, Z) \in Q \times \Gamma^*$, entonces $\delta(q, a, Z) = \emptyset$ para todo $a \in \Sigma$.

Usaremos DPDA (por sus siglas en inglés) como sinónimo de "autómata con pila determinístico".

Observemos que las transiciones de un DPDA tienen las siguientes restricciones:

1. $\mid \delta(q, a, Z) \mid = 0$ o 1.
2. Si $\mid \delta(q, \varepsilon, Z) \mid > 0$, entonces $\mid \delta(q, a, Z) \mid > 0$ para todo $a \in \Sigma$.

Haremos algunos comentarios respecto a los DPDA

1. los lenguajes aceptados por pila vacía correspondientes a DPDA son lenguajes que tienen la siguiente propiedad: si x y y son cadenas aceptadas y $x \neq y$, entonces x no puede ser prefijo de y. En particular, hay lenguajes regulares que no son aceptados por DPDA por pila vacía (por ejemplo 0^*).
2. Consideremos los lenguajes aceptados por DPDA por estado final, en este caso este conjunto de lenguajes contiene estrictamente a los lenguajes regulares y está contenido estrictamente en los lenguajes aceptados por PDA. Por ejemplo, el lenguaje $L = \{aa^R : a \in \{0,1\}^*\}$ es un lenguaje aceptado por un PDA, pero no es posible obtenerlo como lenguaje de un DPDA.