Practica 0

Alumno: Norberto Hernández Cárdenas

Materia: Autómatas y Compiladores

Grupo: 3°

Semestre: 6°

Maestro: Eduardo Cornejo Velázquez

Instituto de Ciencias Básicas e Ingeniería

1 Lenguajes formales

Los lenguajes formales son lenguajes artificiales que se usan para definir reglas en contextos específicos. Se utilizan en matemáticas, informática, lógica y lingüística.

Alfabeto

El alfabeto se trata del conjunto finito de símbolos, como por ejemplo el alfabeto del español que este compuesto por las letras: A,B,C,D,E,F,G,H,I,J,K,L,M,N,Ñ, O,P,Q,R,S,T,U,V,W,X,Y,Z. O en lenguaje con que trabaja una computadora seria: 0,1. Estos son un conjunto finito ya que sus tamaños ya están determinados. Los números naturales tal cual no son un alfabeto, debido a que su conjunto es infinito, si queremos que sean un alfabeto la forma que se tendrían que aplicar es solo considerando los valores del 0 al 9.

Cadenas

Son secuencias ordenadas formadas de elementos de un alfabeto, con el alfabeto español se podría formar las palabras: "HOLA" y "MUNDO", y con el alfabeto binario se podría construir cadenas como "0010" o "1001". Entre las propiedades que tiene una cadena son:

- Longitud: La longitud nos da la cantidad de símbolos que se encuentran dentro de la cadena, donde se cuenta símbolo por símbolo para determinar este valor.
- Apariciones: Las apariciones son la cantidad de veces que aparece un símbolo en especifico dentro de la cadena.
- Ordenación: La ordenación determina en que forma se ordenan las cadenas, el primer factor que se toma en cuenta es la longitud, donde va primero la cadena más corta, en el caso que sean iguales se toma un segundo factor, cada símbolo tiene un orden dentro del alfabeto, donde en el español el primero símbolo es A y el ultimo el Z, entonces al ordenar cadenas de mismo tamaño va primero la cadena que tenga el primer símbolo del alfabeto o más cercano, en el caso que sean iguales comprobarlo con su siguiente símbolo, y así consecutivamente hasta que no sean iguales.
- Combinaciones: Se tratan del número de combinaciones que se pueden lograr con el alfabeto con N símbolos. Si tenemos un alfabeto binario las combinaciones con 2 dígitos seria de 2^2 ósea 4 combinaciones en total, y con 4 dígitos seria 2^4 ósea 16 combinaciones en total.

Clausura de Kleene

Es una forma de representar todas las posibles combinaciones que se pueden lograr mediante un alfabeto en específico, básicamente una forma de representar el infinito de un alfabeto incluyendo es vacío (Σ^*). Si queremos representar estas combinaciones pero sin contar el vacio entonces en lugar de usar el asterisco se usa el símbolo de más Σ^+).

Concatenación

La concatenación es una operación en la cual consiste en combinar dos cadenas

(o palabras) para formar una nueva cadena. Teniendo las cadenas u y v, su concatenación seria $u \cdot v$. Por ejemplo si la concatenamos la cadea "abc" y "def" el resultado seria "abcdef", si concatenamos la cadea "hola" y "mundo" el resultado seria "holamundo". Igual se pueden concatenar cadenas vacías, pero esto no cambiaría la cadena a la que se concatena ($u \cdot \lambda = u$).

Potencia

La potencia es una operación que se aplica a cadenas o lenguajes y representa la repetición de una cadena o lenguaje un número específico de veces. La potencia se define de manera recursiva y depende del exponente que se utilice. Si la potencia es 1 simplemente nos dará la cadena, si es 0 nos devolverá vacío. (Hopcroft, 2002).

$$x^n \begin{cases} \lambda \rightarrow = 0 \\ x^{n-1} \cdot x \rightarrow n > 0 \end{cases}$$

Si tenemos la cadena "abc" y queremos elevarla a la potencia de 3 el resultado seria "abcabcabc" ("abc"^{3} = "$abcabcabc$"), si la cadena es "hola" y la potencia es 4 entonces tendremos "holaholaholahola" ("$hola$"^{4} = "$holaholaholahola$").

Prefijos

Un prefijo de una cadena es cualquier secuencia inicial de símbolos de esa cadena. Dada una cadena w un prefijo de w es cualquier cadena u tal que $w = u \cdot v$, donde v es otra cadena. Si tenemos la cadena "abc" su prefijos son: $\{\lambda, a, ab, abc\}$.

Sufijos

Un sufijo de una cadena es cualquier secuencia final de símbolos de esa cadena. Dada una cadena w, un sufijo de w es cualquier cadena v tal que $w = u \cdot v$, donde u es otra cadena.Si tenemos la cadena "def" su sufijos son: $\{\lambda, f, ef, def\}$.

Segmentos

Un segmento o subcadena de una cadena es cualquier secuencia de símbolos consecutivos dentro de esa cadena. Dada una cadena w, un segmento de w es cualquier cadena u tal que $w = x \cdot u \cdot y$, donde x e y son cadenas.Si tenemos la cadena "abcd" su sufijos son: $\{\lambda, a, b, c, d, ab, bc, cd, abc, bcd, abcd\}$.

Reverso

El **reverso** (también llamado **inversión**) de una cadena es una operación que consiste en invertir el orden de los símbolos que la componen. Dada una cadena w, su reverso se denota como w^R y se obtiene escribiendo los símbolos de w en orden inverso. Si por ejemplo la cadena es "hola" entonces su reverso seria "aloh" y si la cadena fuera "mundo" su reverso seria "odnum". En el caso de que la cadena sea vacía entonces su reverso será vacía, si la cadena está conformada por un solo símbolo entonces la cadena segura estando igual. Dada una cadena $w = a_1 a_2 a_3 \ldots a_n$, donde a_i son símbolos del alfabeto, el reverso de w es:

$$(xa)^r = ax^r$$

Palabras capicúa

Estas son todas las cadenas el cual en su formato original y en su reverso sean del mismo valor, el ejemplo más simple seria una cadea de un solo dígito, en un

caso de una cadena más larga podria ser oso, donde su revero seria oso.

$$x = x^r$$

¿Qué es un lenguaje?
Un lenguaje es un conjunto de cadenas (o palabras) formadas a partir de un alfabeto específico. Estas cadenas están compuestas por símbolos que pertenecen a un conjunto finito y no vacío, llamado **alfabeto**.

$$L \subset \Sigma^*$$

Operaciones booleanas de conjuntos Los lenguajes son conjuntos por lo que estos se pueden operar como estos. Si queremos hacer la unión de 2 lenguajes el resultado que obtendríamos todas las palabras presentes en los dos lenguajes. Como, por ejemplo, si tenemos un lenguaje que posee todas las posibles cadenas que empiecen con A y en otro lenguaje todas las cadenas que empiecen con B la unión serian todas las cadenas que empiecen con A y B.

$$L1 = \{Cx : x \in \Sigma^*\}$$

$$L2 = \{xD : x \in \Sigma^*\}$$

$$L1 \cup L2 = L3$$

La intersección de dos lenguajes serian todas las cadenas que cumplan tanto con la condición del primer lenguaje como la del segundo lenguaje, como por ejemplo, si tenemos un lenguaje que posee todas las posibles cadenas que empiecen con C y en otro lenguaje todas las cadenas que terminen con D la unión serian todas las cadenas que empiecen con C y D.

$$L1 = \{Cx : x \in \Sigma^*\}$$

$$L2 = \{xD : x \in \Sigma^*\}$$

$$L1 \cap L2 = L3$$

Producto de un lenguaje
Se trata de una operación que combina dos lenguajes para formar un nuevo lenguaje. Dados dos lenguajes L_1 y L_2, el producto $L_1 \cdot L_2$ (o simplemente $L_1 L_2$) es el lenguaje que contiene todas las cadenas que se pueden formar concatenando una cadena de L_1 con una cadena de L_2.

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1 \text{ y } w_2 \in L_2\}$$

Como por ejemplo, si tenemos el lenguaje 1 que contiene $\{a, ab\}$ y el lenguaje 2 que contiene $\{c, cd\}$ su producto seria

$$L1 = \{a, ab\}$$

$$L2 = \{c, cd\}$$

$$L_1 \cdot L_2 = \{ac, acd, abc, abcd\}$$

Pero si queremos que trabajar con leguajes infinito en estos casos tenemos que hacer una abstracción del resultado, teniendo un lenguaje 1 donde todas sus cadenas empiecen con A y un lenguaje 2 donde todas sus cadenas terminen con B, el producto de estos dos serian cadenas que empiecen con A y terminen con B.

$$L1 = \{Ax : x \in \Sigma^*\}$$

$$L2 = \{xB : x \in \Sigma^*\}$$

$$L1 \cdot L2 = \{AxB : x \in \Sigma^*\} = \{AB, AAB, ABB, \cdots\}$$

Hay que tener en cuenta que el producto vacío solo va estar presente si se encuentra en los dos lenguajes, en caso contrario no aparecerá, haciendo no posible tener la cadena "AB" siendo que en esta se encuentra el producto vacío entre A y B.

Potencia sobre lenguaje

Es una operación que generaliza la concatenación de un lenguaje consigo mismo un número específico de veces. Dado un lenguaje L y un número entero no negativo n, la potencia L^n se define como la concatenación de L consigo mismo n veces.

$$x^n \begin{cases} \{\lambda\} \text{ si } n = 0 \\ L^{n-1} \cdot L \text{ si } n > 0 \end{cases}$$

Por ejemplo si tenemos un lenguaje finito con las cadenas $\{0, 1\}$ entonces su potencias podrían ser:

- $L^0 = \{\lambda\}$.
- $L^1 = L = \{0, 1\}$.
- $L^2 = L \cdot L = \{00, 01, 10, 11\}$.
- $L^3 = L \cdot L \cdot L = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

Si tenemos un lenguaje finito con las cadenas $\{a, b, c\}$ entonces su potencias podrían ser:

- $L^0 = \{\lambda\}$.
- $L^1 = L = \{a, b, c\}$.
- $L^2 = L \cdot L = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$.

Cierre

Del anterior tema de potencias podemos determinar el cierre, que se trata de la union de todas las potencias de un lenguaje empezando desde L elevado a 0.

$$L^* = \cup L^i$$

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$

Conciente de un lenguaje
El cociente de un lenguaje consiste en removerle a todas las cadenas que contengan una subcadena de símbolos, ya sea del lado izquierdo o derecho.

$$u^{-1}L = \{v \in \Sigma^* : uv \in L\}$$

Por ejemplo si tenemos un lenguaje finito que contiene las cadenas $\{ab, ac, ad\}$ y queremos eliminar "a" el resultado sería:

$$(a)^{-1}L = b, c, d$$

En el caso de que tengamos un lenguaje infinito tenemos que abstractar el resultado, si es un lenguaje donde todas sus cadenas empiecen con "ab" y eliminamos la primera subcadena "ab" de todo el lenguaje el resultado sería:

$$L = \{\text{ab}x : x \in \Sigma^*\}$$

$$(\text{ab})^{-1}L = \{x : x \in \Sigma^*\}$$

Homomorfismo en un lenguaje
Es una función que transforma cada símbolo de un alfabeto en una cadena sobre otro alfabeto. Esta función se extiende naturalmente a cadenas y lenguajes, permitiendo transformar un lenguaje en otro. Los homomorfismos son herramientas fundamentales en la teoría de lenguajes formales, ya que permiten estudiar propiedades de lenguajes y realizar transformaciones sistemáticas.

$$h : \Sigma \to \Gamma^*$$

Como por ejemplo si tenemos un lenguaje 1 que contiene $\{a, b\}$ y un lenguaje 2 que contiene $\{0, 1\}$, su homorfismo se definiria como:

$$h(a) = \text{“01”}, \quad h(b) = \text{“10”}.$$

Si $w =$ “ab”, entonces:

$$h(w) = h(a) \cdot h(b) = \text{“01”} \cdot \text{“10”} = \text{“0110”}.$$

Si $L = \{$“a”, “ab”$\}$, entonces:

$$h(L) = \{\text{“01”}, \text{“0110”}\}.$$

En otro caso, si tenemos un lenguaje 1 que contiene $\{x, y\}$ y un lenguaje 2 que contiene $\{a, b\}$, su homorfismo se definiria como:

$$h(x) = \text{“a”}, \quad h(y) = \text{“bb”}.$$

- Si $w =$ “xyx”, entonces:

$$h(w) = h(x) \cdot h(y) \cdot h(x) = \text{“a”} \cdot \text{“bb”} \cdot \text{“a”} = \text{“abba”}.$$

- Si $L = \{$“x”, “yy”$\}$, entonces:

$$h(L) = \{\text{“a”}, \text{“bbbb”}\}.$$

2 Autómatas

¿Qué son los autómatas?
En el ámbito de la teoría de la computación y los lenguajes formales, un autómata es un modelo matemático abstracto que representa una máquina capaz de procesar cadenas de símbolos y determinar si pertenecen a un lenguaje específico. Los autómatas son fundamentales en el diseño de compiladores, procesadores de texto y sistemas de verificación, entre otras aplicaciones. Los automatas poseen los siguientes elementos:

- Q: Conjunto finito de estados. Son las posibles situaciones en la que una cadena se puede encontrar al ser procesada.
- Σ: Alfabeto de entrada (conjunto finito de símbolos). Son los símbolos con los que el autómata puede trabajar.
- δ: Función de transición, que define cómo el autómata pasa de un estado a otro en función del símbolo de entrada.
- q_0: Estado inicial ($q_0 \in Q$). Es el cual determina el estado desde el cual se comienza el procesamiento de la entrada.
- F: Conjunto de estados finales o de aceptación ($F \subseteq Q$). Los estados finales son los cuales al terminar de procesar una cadena se encuentra en una de estos estados se considerada aceptada.

El autómata procesa una cadena de símbolos y, dependiendo de su configuración, acepta o rechaza la cadena.

Entre los tipos de autómatas que se van a manejar en este reporte se encuentra:

- Automatas finitos deterministas (AFD): Cada transición está determinada únicamente por el estado actual y el símbolo de entrada, no permite entradas vacías.
- Automatas finitos no deterministas (AFN): Puede tener múltiples transiciones para un mismo símbolo desde un estado, este si permite entradas vacías.

Autómata finito determinista
El autómata finito determinista (AFD) es una máquina de estados se encarga de procesar entradas. Es "finito" porque tiene un número limitado de estados, y "determinista" porque, para cada estado y símbolo de entrada, existe exactamente una transición definida. Los AFD son fundamentales en el diseño de compiladores, procesadores de texto y sistemas de verificación. Un autómata finito determinista es una tupla $(Q, \Sigma, \delta, q_0, F)$, donde anteriormente ya determinamos que representa cada uno de estos símbolos. (Kelly,1995).
El funcionamiento de un AFD es el siguiente:

- El autómata comienza en el estado inicial q_0.

- Lee la cadena de entrada símbolo por símbolo.
- Para cada símbolo, utiliza la función de transición δ para moverse al siguiente estado.
- Después de procesar toda la cadena, si el autómata se encuentra en un estado final $q \in F$, la cadena es **aceptada**. De lo contrario, es **rechazada**.

Una de las formas de representar los AFD es mediante tabla de transiciones. Por ejemplo si tenemos el AFD donde se aceptan solo cadenas que empiecen con el símbolo 0 seria:

$$(\{q0, q1, q2\}, \{0, 1\}, \delta, q0, \{q1\})$$

Donde las transiciones son:

$$\delta = \{(q0, 0) = q1, (q0, 1) = q2, (q1, 0) = q1, (q1, 2) = q1, (q2, 0) = q2, (q2, 1) = q2, \}$$

La forma en que se veria en la tabla seria la siguiente:

	0	1
q0	q1	q2
q1	q1	q1
q2	q2	q2

En el anterior caso las transiciones estan completas ya que no hay ningun estado que le falte como procesa algun simbolo, pero en el caso que falte se veria asi:

$$(\{q0, q1, q2\}, \{0, 1\}, \delta, q0, \{q1\})$$

$$\delta = \{(q0, 0) = q1, (q0, 1) = q2, (q1, 0) = q1, (q1, 2) = q1, (q2, 1) = q2\}$$

En la tabla de transiciones:

	0	1
q0	q1	q2
q1	q1	q1
q2	-	q2

Esto representa que en caso de que la cadena procesada se encuentra en el estado q2 y su siguiente símbolo es 0 de inmediatamente la cadena se considera rechazada.

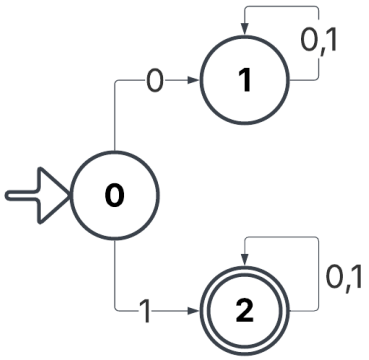
Otra forma de representar los AFD es mediante grafos. Es una representación gráfica donde los estados se representan como nodos, las transiciones con flechas etiquetadas por el símbolo que causa ese cambio de estado, una flecha que apunta al estado donde se empieza y para representar los estados finales los nodos se representan como un círculo encerrado sobre otro. La forma en que se verian

los grafos seria de la siguiente forma. Un autómata que acepta cadenas que empiecen con "1":

$$(\{q0, q1, q2\}, \{0, 1\}, \delta, q0, \{q2\})$$

$$\delta = \{(q0, 0) = q1, (q0, 1) = q2, (q1, 0) = q1, (q1, 2) = q1, (q2, 0) = q2, (q2, 1) = q2\}$$

Y el grafo se veria de la siguiente forma:

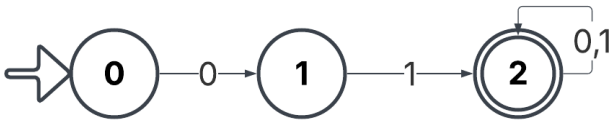


Otro ejemplo donde las transaciones no estan completas, donde se aceptan las cadenas que empiecen con la subcadena "01":

$$(\{q0, q1, q2\}, \{0, 1\}, \delta, q0, \{q2\})$$

$$\delta = \{(q0, 0) = q1, (q1, 1) = q2, (q2, 0) = q2, (q2, 1) = q2\}$$

Y el grafo se veria de la siguiente forma:



Como se puede apreciar el cambio de estado se representa como una flecha indicando a otro estado, y si un estado no cambia de estado en la transición este se representa como una flecha apuntado al mismo estado, si hay una transición faltante esta solamente no se representa en el grafo.

Autómata finito no determinista

El autómata finito no determinista (AFN) es igual una máquina de estados que se encarga de procesar entradas de cadenas, comparte los mismos elementos que su contraparte AFD donde es "finito" porque tiene un n´umero limitado

de estados, su tupla $(Q, \Sigma, \delta, q_0, F)$ es similar a la de un AFD, en lo que se diferencia es que este es "No determinista", un AFN puede tener múltiples transiciones para un mismo símbolo desde un estado, e incluso puede realizar transiciones sin consumir un símbolo (transiciones vacías o ϵ-transiciones).

$$\delta \text{ (AFD)} : Q \times \Sigma \to Q$$

$$\delta \text{ (AFN)} : Q \times \Sigma \to 2^Q$$

Cuando se representa su función de transición este puede poseer multiples cambios de estados causado por un mismo símbolo. Pero ahora ¿Como es que funcionan estos autómatas?

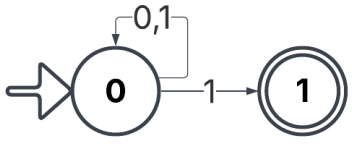
- El autómata comienza en el estado inicial q_0.
- Lee la cadena de entrada símbolo por símbolo.
- Para cada símbolo, el autómata puede:
  - Moverse a cualquier estado definido por la función de transición δ.
  - Realizar transiciones vacías (ϵ-transiciones) sin consumir un símbolo.
- Después de procesar toda la cadena, si el autómata se encuentra en al menos un estado final $q \in F$, la cadena es **aceptada**. De lo contrario, es **rechazada**.

Su representación sigue siendo igual que la de un AFD, aqui un ejemplo de como se verian:

$$(\{q0, q1\}, \{0, 1\}, \delta, q0, \{q1\})$$

$$\delta = \{(q0, 0) = q0, (q0, 1) = q1, (q1, 1) = q0\}$$

	0	1
q0	$\{q0\}$	$\{q0, q1\}$
q1	-	-

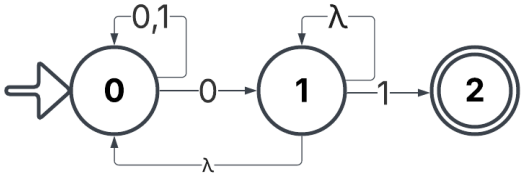


Otro ejemplo seria:

$$(\{q0, q1, q2\}, \{0, 1\}, \delta, q0, \{q1\})$$

$$\delta = \{(q0, 0) = q0, (q0, 0) = q1, (q0, 1) = q0, (q1, \lambda) = q1, (q1, \lambda) = q0, (q1, 1) = q2\}$$

	0	1	λ
q0	$\{q0\}$	$\{q0, q1\}$	-
q1		$\{q2\}$	$\{q0, q1\}$
q2	-	-	-



Convertir un AFN a un AFD
La transformación de un autómata finito no determinista a un autómata finito determinista es un proceso fundamental en la teoría de lenguajes formales. Este procedimiento garantiza que, dado un AFN, se pueda construir un AFD equivalente que reconozca el mismo lenguaje. A continuación se explicara los pasos que se deben de seguir.

1. **Definir los componentes del Automata**:
   - La tupla de nuestro AFD seria $(Q, \Sigma, \delta, q_0, F)$
   - Los estados del AFD serán los mismos que el del AFN
   - El alfabeto de entrada Σ será el mismo que el del AFN.
   - La función de transición del AFD se construirá a partir de la función de transición del AFN.
   - El estado inicial del AFD será el del estado inicial del AFN.
   - Los estados finales del AFD serán los mismos que contengan el AFN.
2. **Elaborar la tabla de transición**:
   - Se crea una tabla donde la parte superior este el alfabeto del AFN y del lado izquierdo se estaran poniendo todos los estados posibles de nuestro AFN empezando con el estado inicial.
   - Se obtienen todos los estados a los que cambia el estado inicial con todo el alfabeto, si una de estas transiciones termina en varios estados se juntan en un conjunto y se toman como un solo estado individual.

- Luego de los estados que resultaron se toma uno de estos y se realiza lo mismo, se sacan sus transiciones con el alfabeto. Este proceso se sigue repitiendo hasta que se hayan abordado todos los posibles estados que genera el AFN.

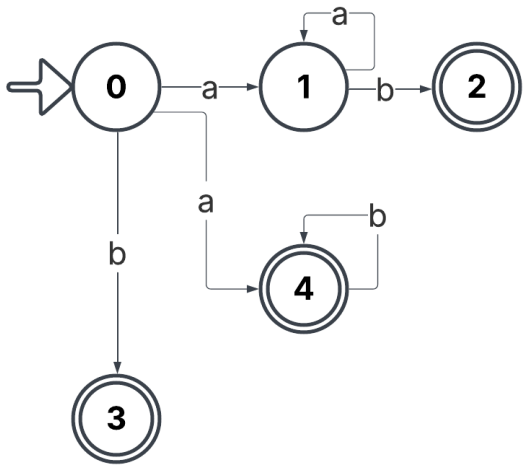
3. **Elaboración de las transiciones del AFD**:

   - Se consideraran los estados que se consiguieron (incluyendo a los conjuntos de estados) y se consideraran como los estados del AFD.
   - Estos estados se van a considerar por el momento "no finales".
   - Basandonos en la tabla de transiciones que se elaboro en el paso anterior es donde se van a tomar las nuevas transiciones del AFD (δ)

4. **Identificar los estados finales del AFD**:

   - Se consideraran estados finales aquellos que anteriormente en el AFN eran estados finales o en el caso que sean unn conjunto de estados contengan un estado que era final anteriormente.

Ahora ya sabiendo cual es el proceso para pasar un AFN a un AFD toca aplicarlo en un ejemplo. Teniendo el siguiente AFN, deseamos pasarlo a un AFD:



$$AFN = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q3, q4\})$$

$$\delta = \{(q0, a) = q1, (q0, a) = q4, (q0, b) = q3, (q1, a) = q1, (q1, b) = q2, (q4, b) = q4\}$$

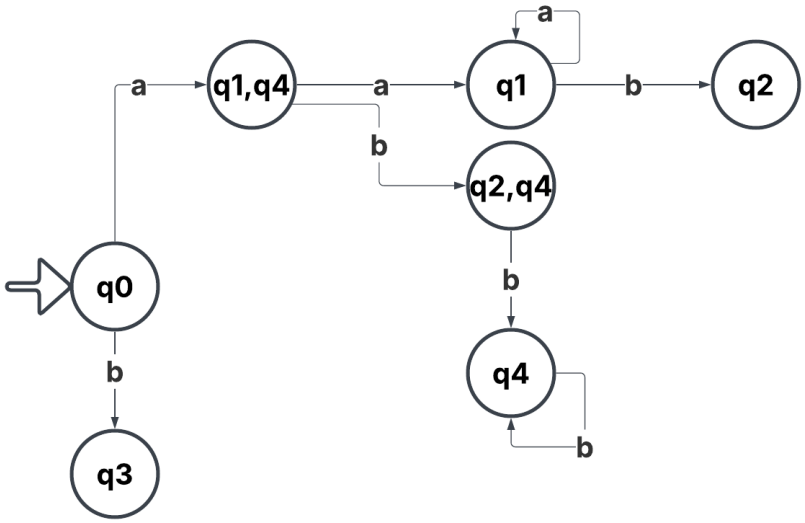
1. **Definir los componentes del Automata**:

$$AFD = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q3, q4\}) = AFN$$

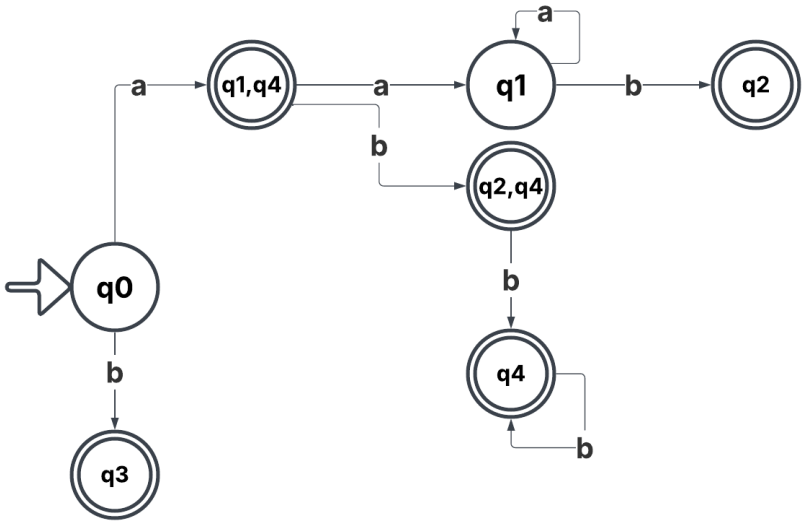
2. **Elaborar la tabla de transición**: Terminamos teniendo con 7 estados diferentes, por ende nuestra tabla de transición se veria asi:

	a	b
$q0$	$\{q1, q4\}$	$\{q3\}$
$\{q1, q4\}$	$\{q1\}$	$\{q2, q4\}$
$\{q3\}$	-	-
$\{q1\}$	$\{q1\}$	$\{q2\}$
$\{q2, q4\}$	-	$\{q4\}$
$\{q2\}$	-	-
$\{q4\}$	-	$\{q4\}$

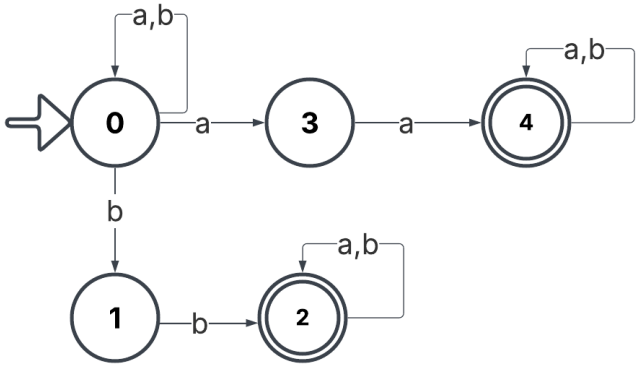
3. **Elaborar la tabla de transición**: Ahora con la tabla podemos elaborar nuestro grafo



4. **Identificar los estados finales del AFD**: Ahora con el conjunto de estados finales podemos determinar que nodos son finales, si el conjunto posee por al menos un estado final ya se puede considerar como uno. $AFN = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q3, q4\})$



Otro ejemplo de AFN a AFD



$$AFN = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q4\})$$

$$\delta = \{(q0, a) = q0, (q0, a) = q3, (q0, b) = q1, (q0, b) = q0, (q1, b) = q2, (q2, a) = q2, (q2, b) = q2,$$
$$(q3, a) = q4, (q4, a) = q4, (q4, ab) = q4\}$$

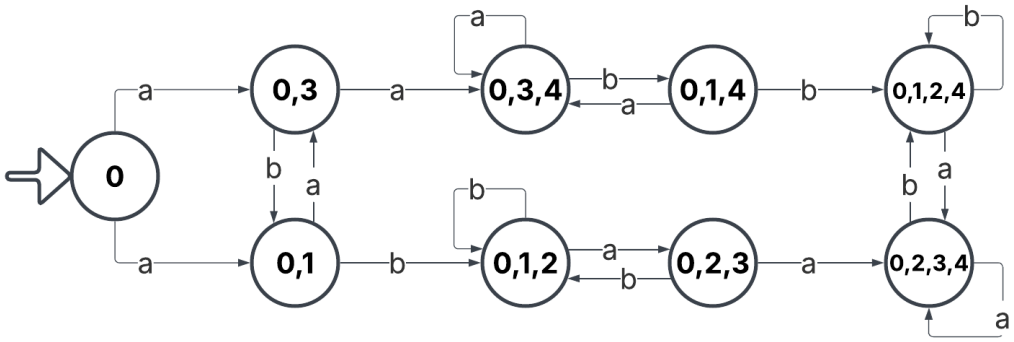
1. **Definir los componentes del Automata**:

$$AFD = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q4\}) = AFN$$

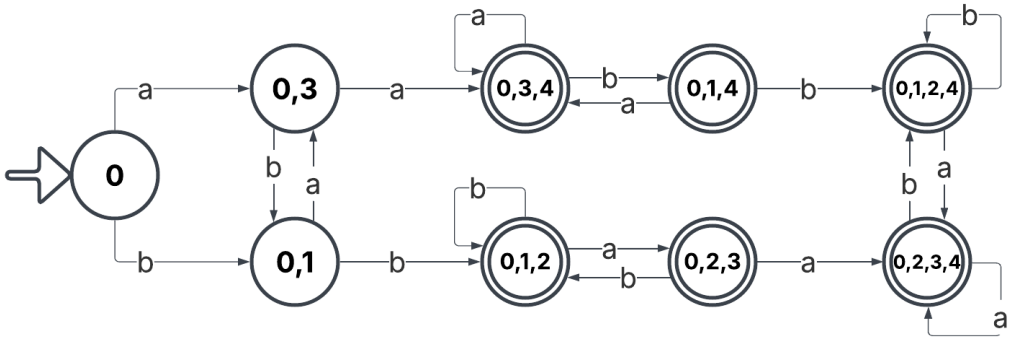
2. **Elaborar la tabla de transición**: Terminamos teniendo con 9 estados diferentes, por ende nuestra tabla de transición se veria asi:

	a	b
$q0$	$\{q0, q3\}$	$\{q0, q1\}$
$\{q0, q3\}$	$\{q0, q3, q4\}$	$\{q0, q1\}$
$\{q0, q1\}$	$\{q0, q3\}$	$\{q0, q1, q2\}$
$\{q0, q3, q4\}$	$\{q0, q3, q4\}$	$\{q0, q1, q4\}$
$\{q0, q1, q2\}$	$\{q0, q2, q3\}$	$\{q0, q1, q2\}$
$\{q0, q1, q4\}$	$\{q0, q3, q4\}$	$\{q0, q1, q2, q4\}$
$\{q0, q2, q3\}$	$\{q0, q2, q3, q4\}$	$\{q0, q1, q2\}$
$\{q0, q1, q2, q4\}$	$\{q0, q2, q3, q4\}$	$\{q0, q1, q2, q4\}$
$\{q0, q2, q3, q4\}$	$\{q0, q2, q3, q4\}$	$\{q0, q1, q2, q4\}$

3. **Elaborar la tabla de transición**: Ahora con la tabla podemos elaborar nuestro grafo



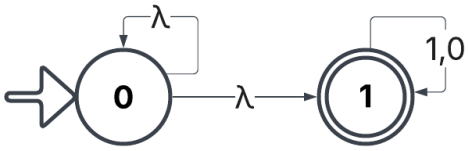
4. **Identificar los estados finales del AFD**: Ahora con el conjunto de estados finales podemos determinar que nodos son finales, si el conjunto posee por al menos un estado final ya se puede considerar como uno. $AFN = \{q0, q1, q2, q3, q4\}, \{a, b\}, \delta, q0, \{q2, q4\})$



3 Autómatas con Transiciones Epsilon

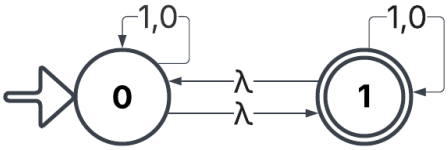
¿Qué es la cadena vacía?
Las cadenas vacias o lambda λ es un símbolo que representa la ausencia de símbolos o una cadena de caracteres de tamaño 0, pero la cadena vacía no representa "null" o un conjunto vacío.
¿Qué es un autómata con transiciones epsilon?
Es una extensión de los autómatas finitos no deterministas (AFN) que permite transiciones sin consumir un símbolo de entrada. Estas transiciones, llamadas transiciones épsilon, permiten al autómata cambiar de estado sin leer ningún símbolo de la cadena de entrada. Esto añade mayor flexibilidad al diseño de autómatas, aunque al final siempre es posible convertirlos en un autómata finito determinista (AFD) equivalente. Como por ejemplo si tenemos un AFN como el siguiente:

$$AFN = \{q0, q1\}, \{a, b\}, \delta, q0, \{q1\})$$
$$\delta = \{(q0, \lambda) = q0, (q0, \lambda) = q1, (q1, 0) = q1, (q1, 1) = q1\}$$



Como se puede apreciar en el ejemplo, se puede hacer uso de una cadena vacía para cambiar al siguiente estado en un AFN. Otro ejemplo haciendo uso del mismo concepto:

$$AFN = \{q0, q1\}, \{a, b\}, \delta, q0, \{q1\})$$
$$\delta = \{(q0, \lambda) = q1, (q0, 0) = q0, (q0, 1) = q0, (q1, \lambda) = q0, (q1, 0) = q1, (q1, 1) = q1\}$$



En este ejemplo se hace uso de la cadena vacía para alternar entre dos estados del AFN.
Convertir un AFND con transiciones epsilon a un AFN
La conversión de un autómata finito no determinista con transiciones épsilon a un autómata finito no determinista sin transiciones épsilon es un paso importante en la teoría de lenguajes formales. Este proceso elimina las

transiciones épsilon y construye un AFN equivalente que reconoce el mismo lenguaje. Un concepto que hay que tener en cuenta es la $\lambda - clausura$ de un estado q es equivalente a ese mismo estado q más todos los estados accesibles mediante λ. A continuación, se describe el procedimiento paso a paso.

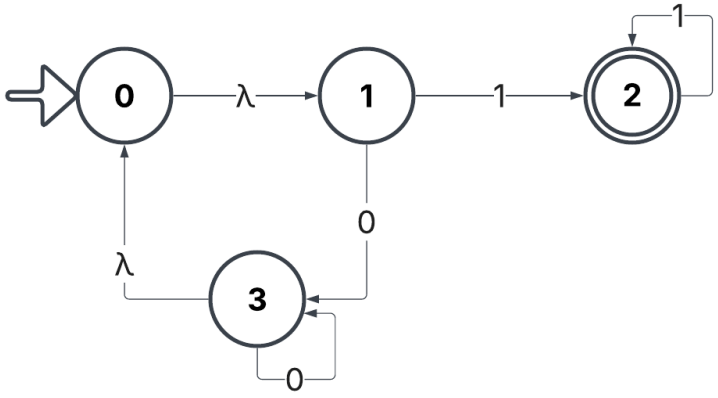
1. **Obtener la $\lambda - clausura$ de cada uno de los estados**:
   - Se obtiene la $\lambda - clausura$ del estado inicial, el cual como se explicó se obtiene de tomando en cuenta el estado actual más todos los estados que se pueden alcanzar mediante el uso la cadena vacía.
   - El mismo procedimiento se repite para todos los estados del autómata.
2. **Construir una tabla de transiciones**:
   - En la tabla en su parte superior vamos a tener el lenguaje de la AFN, y del lado izquierdo se van a poner todos los estados del autómata.
   - En el primer estado de la tabla comprobamos cuales son los estados los que llevan cada símbolo del lenguaje, pero esto lo hacemos con todos los estados que se encuentran en su $\lambda - clausura$.
     - Se toma el primer estado de la $\lambda - clausura$.
     - Si este estado tiene una transición con el símbolo actual a algun estado entonces se agrega al conjunto de estado.
     - Antes de pasar al siguiente estado de la $\lambda - clausura$ se checa que si el anterior estado agregado al conjunto si tiene alguna transición de λ. Si es que lo tiene se agrega el estado al que lleva al conjunto. Se repite este mismo paso con el estado agregado hasta que no agregue estados nuevos al conjunto.
     - Una vez acabado con lo anterior se pasa al siguiente estado de la $\lambda - clausura$ y se vuelve a repetir este proceso.
   - Una vez terminado de hacer todo el proceso con las $\lambda - clausura$ del primer estado se pasa al siguiente y se hace lo mismo con sus propias $\lambda - clausura$.
3. **Elaborar las transiciones del AFN**:
   - Se consideraran los estados del AFN los mismos que tiene el AFN con transaciones epsilon.
   - Estos estados se van a considerar por el momento "no finales".
   - Basandonos en la tabla de transiciones se determinan desde que estado se va a pasar basandonos en la símbolos del lenguaje, como es un AFN un solo símbolo puede tener multiples transiciones.
4. **Identificar los estados finales del AFN**:

- Se consideraran estados finales aquellos que anteriormente en el AFN con transaciones epsilon eran estados finales.
- En caso de que el estado inicial no sea final y dentro de su $\lambda - clausura$ haya un estado final entonces se considera final. estados contengan un estado que era final anteriormente.

A continuación se dará un ejemplo de una conversión de un AFN de transiciones épsilon aun AFN.

$$AFN = \{q0, q1, q2, q3\}, \{0, 1\}, \delta, q0, \{q2\})$$

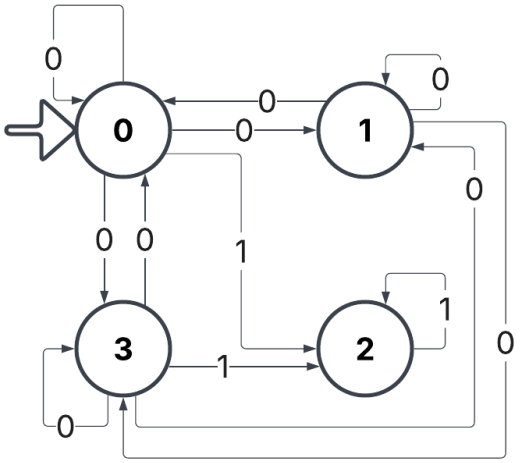
$$\delta = \{(q0, \lambda) = q1, (q1, 0) = q3, (q1, 1) = q2, (q2, 1) = q2, (q3, \lambda) = q0, (q3, 0) = q3\}$$



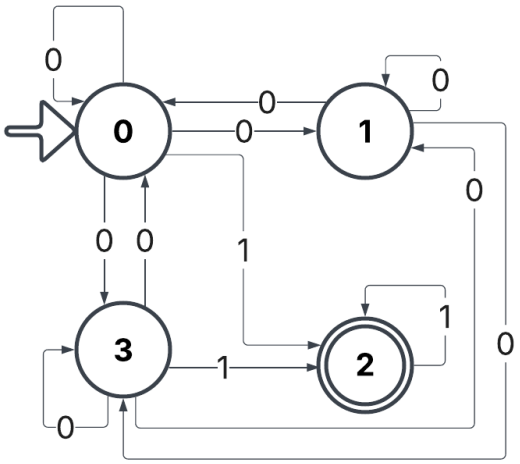
1. **Obtener la $\lambda - clausura$ de cada uno de los estados**:
   - $(\lambda - clausura(q0) = \{q0, q1\}$
   - $(\lambda - clausura(q1) = \{q1\}$
   - $(\lambda - clausura(q2) = \{q2\}$
   - $(\lambda - clausura(q3) = \{q0, q1, q3\}$
2. **Construir una tabla de transiciones**:

	0	1
$q0$	$\{q0, q1, q3\}$	$\{q2\}$
$q1$	$\{q0, q1, q3\}$	$\{q2\}$
$q2$	-	$\{q2\}$
$q3$	$\{q0, q1, q3\}$	-

3. **Elaborar las transiciones del AFN**:



4. **Identificar los estados finales del AFN**:

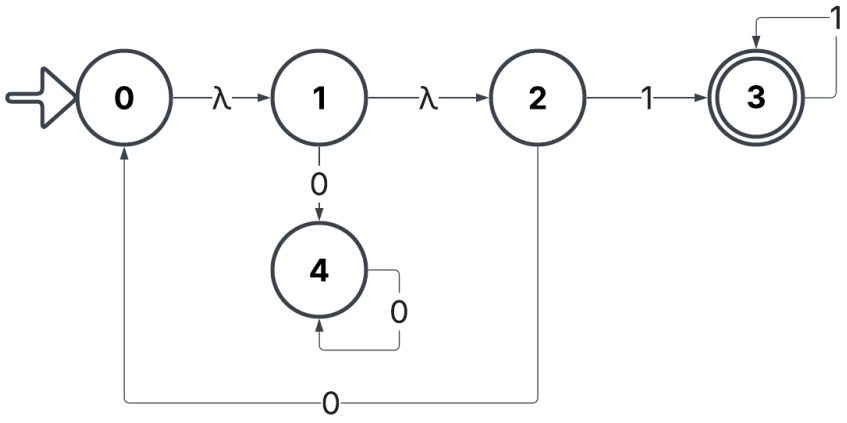


Otro ejemplo de una conversión de un AFN de transiciones épsilon aun AFN:

$$AFN = \{q0, q1, q2, q3, q4\}, \{0, 1\}, \delta, q0, \{q3\})$$

$$\delta = \{(q0, \lambda) = q1, (q1, \lambda) = q2, (q1, 0) = q4, (q2, \lambda) = q0, (q2, 1) = q3, (q3, 1) = q3$$

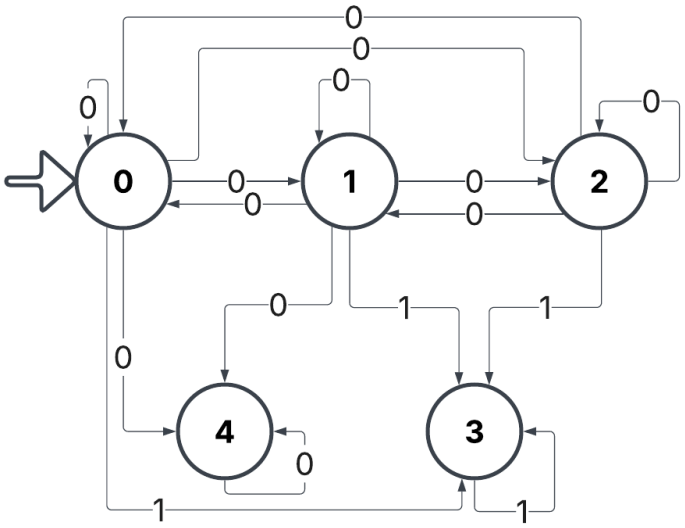
$$, (q4, 0) = q4\}$$



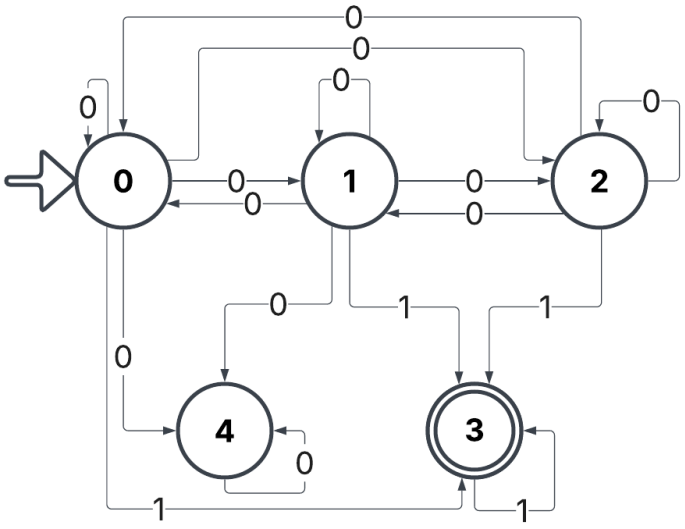
1. **Obtener la $\lambda - clausura$ de cada uno de los estados**:
   - $(\lambda - clausura(q0) = \{q0, q1, q2\}$
   - $(\lambda - clausura(q1) = \{q1, q2\}$
   - $(\lambda - clausura(q2) = \{q2\}$
   - $(\lambda - clausura(q3) = \{q3\}$
   - $(\lambda - clausura(q4) = \{q4\}$
2. **Construir una tabla de transiciones**:

	0	1
$q0$	$\{q0, q1, q2, q4\}$	$\{q3\}$
$q1$	$\{q0, q1, q2, q4\}$	$\{q3\}$
$q2$	$\{q0, q1, q2\}$	$\{q3\}$
$q3$	-	$\{q3\}$
$q4$	$\{q4\}$	-

3. **Elaborar las transiciones del AFN**:



4. **Identificar los estados finales del AFN**:



4 Pattern Matching

¿Qué es el pattern matching?
El pattern matching (coincidencia de patrones) es una técnica utilizada para buscar patrones específicos dentro de una cadena de texto. Los autómatas,

especialmente los autómatas finitos deterministas y autómatas finitos no deterministas, son herramientas fundamentales para implementar esta técnica de manera eficiente. A continuación, se explica cómo funciona el pattern matching con autómatas y su aplicación en la búsqueda de patrones.
Un patrón es una secuencia de caracteres que se desea buscar en una cadena de texto. Por ejemplo, el patrón "abc" busca la secuencia exacta de caracteres "a", "b" y "c" en ese orden.

Algoritmo ingenuo

El algoritmo de búsqueda ingenua (también conocido como algoritmo de fuerza bruta) es un método simple para buscar un patrón dentro de un texto. Aunque no es el más eficiente en términos de complejidad temporal, es fácil de entender e implementar. Este algoritmo compara el patrón con todas las posibles subcadenas del texto, una por una, hasta encontrar una coincidencia o hasta que se hayan examinado todas las subcadenas. El procedimiento del algoritmo se explicara a continuación:

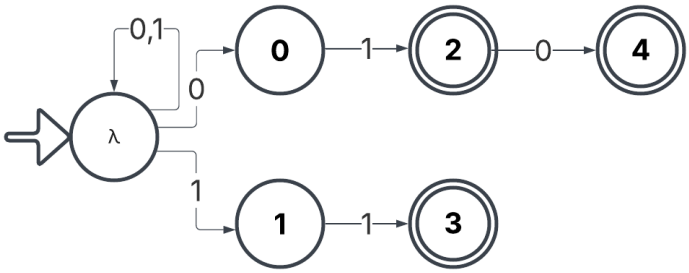
1. El algoritmo recibe una cadena de texto como entrada.
2. Tenemos un conjunto de patrones C que se desean buscar dentro del texto de entrada.
3. Se construye un árbol aceptor de prefijos basandonos en el conjunto de patrones C. Este árbol es un automata el cual su tupla $(Q, \Sigma, \delta, q0, F)$ se encuentra definida como:
   - Q: Esta compuesto por todos los prefijos que nos podemos encontrar en el conjunto de patrones C.
   - Σ: Esta conformada por los simbolos que se pueden encontrar en los patrones.
   - $q0$: Representa la cadena vacía.
   - F: Son todos los estados finales que representan una patron completo del conjunto de patrones C.
   - δ: Define las transiciones entre estados para cada símbolo que lleven hacia una cadena del conjunto de patrones C.

     $$\delta(x, s) = xs \text{ si } xs \in Q$$
4. Una vez formado el árbol lo único que faltaría es agregar al estado inicial una transición con todos los símbolos del alfabeto que lleven al estado inicial, básicamente un bucle, asi volviendo nuestro autómata en uno no determinista.
   - La razón de porque se hace esto es por cada posición de la cadena de entrada se empiece un nuevo procesamiento que recorra el arbol.
   - De esta forma cada vez que uno de los procesos llegue a un estado final se pueda notificar y estimar su ubicación y las veces que se encontró una subcadena dentro de la cadena de entrada.

A continuación un ejemplo de como se ve el algoritmo ingenuo en acción. Tenemos nuestro autonoma finito no determinista $(Q, \Sigma, \delta, q0, F)$, el cual se encuentra definida como:

- $Q = \{01, 010, 11\}$
- $\Sigma = \{0, 1\}$
- $q0 = \lambda$
- $F = \{q2, q3, q4\}$
- $\delta = \{(\lambda, 0) = \lambda, (\lambda, 0) = q0, (\lambda, 1) = \lambda, (\lambda, 1) = q1, (q0, 1) = q2, (q1, 1) = q3, (q2, 0) = q4\}$

	0	1
λ	$\{\lambda, q0\}$	$\{\lambda, q1\}$
q0	-	$\{q2\}$
q1	-	$\{q3\}$
q2	$\{q4\}$	-
q3	-	-
q4	-	-



La cadena de entrada es la siguiente: "0110101"

- $0 \rightarrow \lambda, q0, \boxed{\text{q2}}$
- $1 \rightarrow \lambda, q1, \boxed{\text{q3}}$
- $1 \rightarrow \lambda, q1$
- $0 \rightarrow \lambda, q0, \boxed{\text{q2}}, \boxed{\text{q4}}$
- $1 \rightarrow \lambda, q1$
- $0 \rightarrow \lambda, q0, \boxed{\text{q2}}$

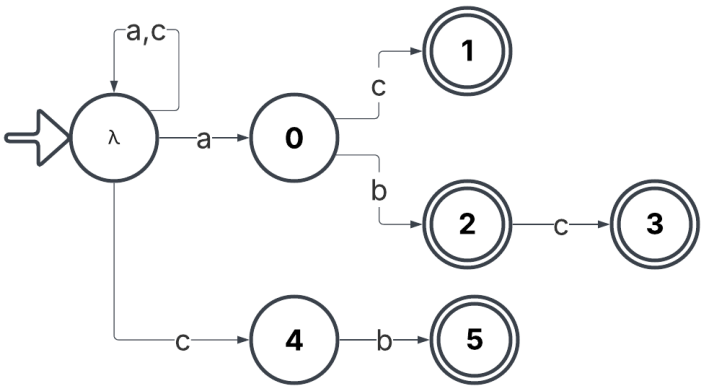
- $1 \rightarrow \lambda, q1$

Se encontro la cadena "01" 3 veces en total en las posiciones $\{1, 4, 6\}$, se encontro la cadena "010" 1 vez en la posición $\{5\}$ y se encontro la cadena "11" 1 vez en la posición $\{2\}$.

Otro ejemplo. Tenemos nuestro autonoma finito no determinista $(Q, \Sigma, \delta, q0, F)$, el cual se encuentra definida como:

- $Q = \{ac, ab, abc, cb\}$
- $\Sigma = \{a, b, c\}$
- $q0 = \lambda$
- $F = \{q1, q2, q3, q5\}$
- $\delta = \{(\lambda, a) = \lambda, (\lambda, c) = \lambda, (\lambda, a) = q0, (\lambda, c) = q4, (q0, b) = q1, (q0, c) = q2, (q2, c) = q3, (q4, b) = q5\}$

	a	b	c
λ	$\{\lambda, q0\}$	-	$\{\lambda, q4\}$
q0	-	$\{q1\}$	$\{q2\}$
q1	-	-	-
q2	-	-	$\{q3\}$
q3	-	-	-
q4	-	$\{q5\}$	-
q5	-	-	-



La cadena de entrada es la siguiente: "abcabcbabc"

- $a \rightarrow \lambda, q0, \boxed{\text{q2}}, \boxed{\text{q3}}$
- $b \rightarrow \lambda$

- $c \rightarrow \lambda, q4$
- $a \rightarrow \lambda, q0, \boxed{\text{q2}}, \boxed{\text{q3}}$
- $b \rightarrow \lambda$
- $c \rightarrow \lambda, q4, \boxed{\text{q5}}$
- $b \rightarrow \lambda$
- $a \rightarrow \lambda, q0, \boxed{\text{q2}}, \boxed{\text{q3}}$
- $b \rightarrow \lambda$
- $c \rightarrow \lambda, q4$

Se encontro la cadena "ab" 3 veces en total en las posiciones $\{1, 4, 8\}$, se encontro la cadena "abc" 3 veces en las posiciones $\{2, 5, 9\}$ y se encontro la cadena "cb" 1 vez en la posición $\{6\}$.
Es un algoritmo bastante seccillo pero no muy eficiente de implementar ya que solo funciona si las subcadenas se encuentran lo más rapido posible, en el caso de que esten muy lejos del inicio o incluso si no estan este proceso se vuelve muy tardado.

Autómata diccionario

Los autómatas diccionarios son una extensión de los autómatas finitos diseñados para reconocer múltiples patrones simultáneamente. Estos autómatas son especialmente útiles en aplicaciones donde se necesita buscar varias palabras clave en un texto, como en motores de búsqueda, filtrado de contenido o análisis léxico. El autómata más común utilizado para este propósito es el autómata de Aho-Corasick, que permite realizar búsquedas eficientes de múltiples patrones en tiempo lineal respecto al tamaño del texto.
El procedimiento del algoritmo se explicara a continuación:

1. El algoritmo recibe una cadena de texto como entrada.
2. Tenemos un conjunto de patrones C que se desean buscar dentro del texto de entrada.
3. Se construye un árbol aceptor de prefijos basandonos en el conjunto de patrones C. Este árbol es un automata el cual su tupla $(Q, \Sigma, \delta, q0, F)$ se encuentra definida como:
    - Q: Esta compuesto por todos los prefijos que nos podemos encontrar en el conjunto de patrones C.
    - Σ: Esta conformada por los simbolos que se pueden encontrar en los patrones.
    - $q0$: Representa la cadena vacía.

- F: Estara formado por los estados que estén identificados con una palabra que tenga, al menos, un sufijo en C.

$$F = Pref(C) \cap \Sigma^* C$$

- δ: En cada estado debe haber una transición definida para cada símbolo del alfabeto.

$$\delta(x, a) = h(xa)$$

  - Si en alguno de los estado recibe un símbolo por el cual no puede lleva a alguno de los prefijos de C entonces se hace una transición que lleve al subfijo más largo que si tenga un estado asignado.
  - La forma en que se deben encontrar estas transiciones es que se deben anotar en que subcadenas hacen aparición otras subcadenas

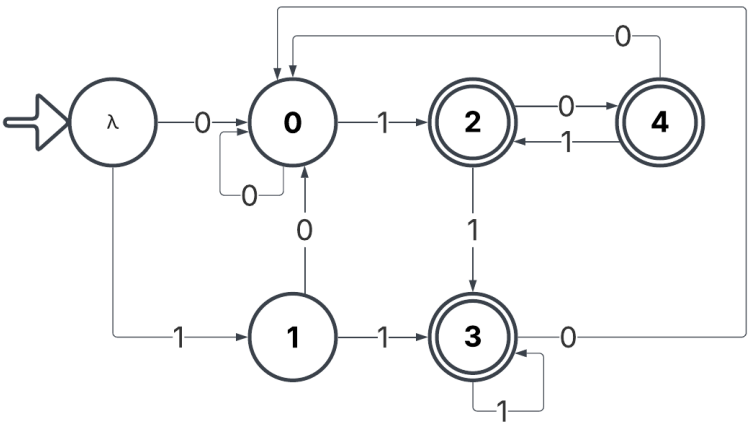
A continuación un ejemplo de como se ve el algoritmo ingenuo en acción. Tenemos nuestro autonoma finito no determinista $(Q, \Sigma, \delta, q0, F)$, el cual se encuentra definida como:

- $Q = \{p1 = 01, p2 = 010, p3 = 11\}$
- $\Sigma = \{0, 1\}$
- $q0 = \lambda$
- $F = \{q2, q3, q4\}$

Estados	{01}	{010}	{11}
Patrones	p1	p1,p2	p3

- $\delta = \{(\lambda, 0) = q0, (\lambda, 1) = q1, (q0, 0) = q0, (q0, 1) = q2, (q1, 0) = q0, (q1, 1) = q3, (q2, 0) = q4, (q2, 1) = q3, (q3, 0) = q0, (q3, 1) = q3, (q4, 0) = q0, (q4, 1) = q2\}$

	0	1
λ	q0	q1
q0	q0	q2
q1	q0	q3
q2	q4	q3
q3	q0	q3
q4	q0	q2



La cadena de entrada es la siguiente: "0110101"

- $0 \rightarrow q0$
- $1 \rightarrow$ q2
- $1 \rightarrow$ q3
- $0 \rightarrow q0$
- $1 \rightarrow$ q2
- $0 \rightarrow$ q4
- $1 \rightarrow$ q2

Se encontro la cadena "01" 3 veces en total en las posiciones $\{1, 4, 6\}$, se encontro la cadena "010" 1 vez en la posición $\{5\}$ y se encontro la cadena "11" 1 vez en la posición $\{2\}$.

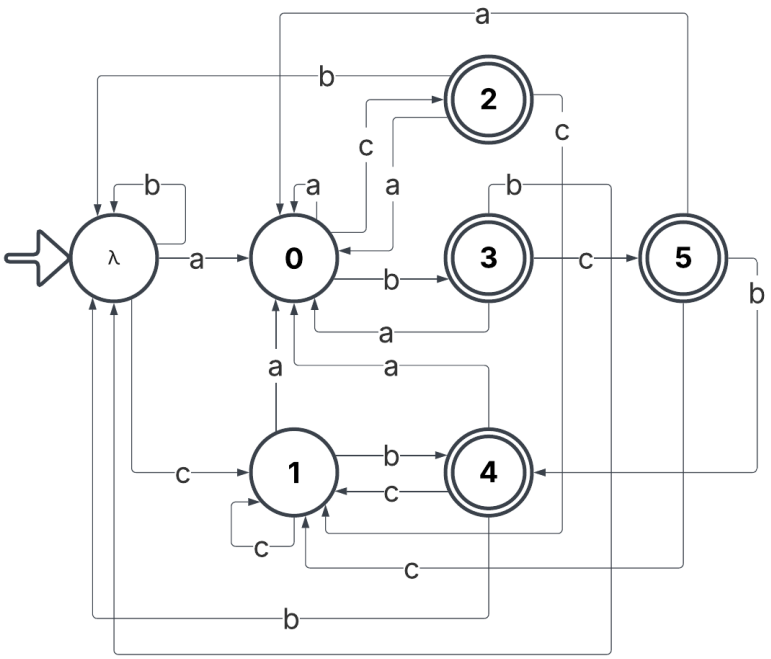
Otro ejemplo. Tenemos nuestro autonoma finito no determinista $(Q, \Sigma, \delta, q0, F)$, el cual se encuentra definida como:

- $Q = \{p1 = ac, p2 = ab, p3 = abc, p4 = cb\}$
- $\Sigma = \{a, b, c\}$
- $q0 = \lambda$
- $F = \{q2, q3, q4, q5\}$

Estados	$\{ac\}$	$\{ab\}$	$\{abc\}$	$\{cb\}$
Patrones	p1	p2	p2,p3	p4

- $\delta = \{(\lambda, a) = q0, (\lambda, b) = \lambda, (\lambda, c) = q1, (q0, a) = q0, (q0, b) = q3, (q0, c) = q2, (q1, a) = q0, (q1, b) = q4, (q1, c) = q1, (q2, a) = q1, (q2, b) = \lambda, (q2, c) = q1, (q3, a) = q0, (q3, b) = \lambda, (q3, c) = q5, (q4, a) = q0, (q4, b) = \lambda, (q4, c) = q1, (q5, a) = q0, (q0, b) = q4, (q0, c) = q1\}$

	a	b	c
λ	q0	λ	q1
q0	q0	q3	q2
q1	q0	q4	q1
q2	q0	λ	q1
q3	q0	λ	q5
q4	q0	λ	q1
q5	q0	q4	q1



La cadena de entrada es la siguiente: "abcabcbabc"

- $a \rightarrow q0$
- $b \rightarrow \boxed{\text{q3}}$
- $c \rightarrow \boxed{\text{q5}}$
- $a \rightarrow q0$

- $b \rightarrow \boxed{\text{q3}}$
- $c \rightarrow \boxed{\text{q5}}$
- $b \rightarrow q4$
- $a \rightarrow q0$
- $b \rightarrow \boxed{\text{q3}}$
- $c \rightarrow \boxed{\text{q5}}$

Se encontro la cadena "ab" 3 veces en total en las posiciones $\{1, 4, 8\}$, se encontro la cadena "abc" 3 veces en las posiciones $\{2, 5, 9\}$ y se encontro la cadena "cb" 1 vez en la posición $\{6\}$.

5 Clases de equivalencia

Lenguaje por la derecha
Dado un autómata finido determinista $(Q, \Sigma, \delta, q0, F)$ el lenguaje por la derecha (R) de un de los estados de su Q es el conjunto de todas cadenas que pueden llevar a este mismo estado.
El lenguaje L es todo el conjunto de todas las cadenas que acepta el AFD como valido. Si el AFD tiene su estado inical como final entonces se puede considerar su $R_{q0} = L$.
El el lenguaje por la derecha nos permite simplificar nuestros AFD ya que con este podemos encontrar los estados que tienen un comportamiento similar y de esta forma podemos quitar uno de los dos estados para simplificar el autómata. Si todos los estados del AFD tiene comportamientos diferentes entonces no hay nada que reducir.
Relaciones de equivalencia
Una relación (R) sobre un conjunto de estados (A) es una relación de equivalencia si cumple 3 condiciones:

- Reflexiva: Cada estado tiene el mismo comportamiento que si mismo.

$$\forall x \in A(xRz)$$

- Simétrica: Si un estado x tiene un mismo comportamiento que el estado y, entonces el estado y tiene un mismo comportamiento que el estado x.

$$\forall x, y \in A(xRy \rightarrow yRx)$$

- Transitiva: Si un estado x su comportamiento es igual a un estado y y el estado y su comportamiento es igual a un estado z, entonces esl comportamiento del estado x es el mismo que el del estado z.

$$\forall x, y, z \in A(xRy \wedge yRz \rightarrow aRz)$$

Clases de equivalencia
Las clases de equivalencia son una herramienta fundamental para analizar y clasificar estados o cadenas en función de su comportamiento dentro de un autómata o un lenguaje. Estas clases permiten agrupar elementos que son indistinguibles bajo ciertas condiciones, lo que facilita la simplificación de autómatas y la comprensión de la estructura de los lenguajes.
Dada una relación de quivalencia (R) sobre A y dado un elemento $x \in A$, el conjunto $\{y \in A | xRy\}$ se dice clase de equivalencia de x y se representa por $[x]_r$. El conjunto de sus diferentes clases de equivalencia de R se le llama "Conjunto conciente" de A por R (A/R, y el numero de clases diferentes representa su indice.
Lenguajes regulares
Son aquellos que tiene un numero finito de clases de equivalencias, se pueden representar con un numero finito de estados.
Equivalencia de nerode
Un lenguaje sera regular si tiene un numero finito de clases de equivalencia.

$$x \equiv R_L y \leftrightarrow x^{-1}L = y^{-1}L$$

6 Lenguajes regulares y no regulares

Que un lenguaje es regular significa que tiene un numero finito de clases de equivalencia, osa que se puede representar mediante un autómata finito.
Teorema de Myhill-Nerode
El teorema de Myhill-Nerode es un resultado fundamental en la teoría de lenguajes formales que proporciona una caracterización de los lenguajes regulares en términos de clases de equivalencia. Este teorema no solo permite demostrar si un lenguaje es regular, sino que también proporciona una manera de determinar el número mínimo de estados necesarios para construir un autómata finito determinista que reconozca el lenguaje.
Pasos para aplicar el teorema de Myhill-Nerode

1. **Definir la relación $\equiv_L$**:
    - Para un lenguaje L, identificar las cadenas que son equivalentes bajo $\equiv_L$.
2. **Encontrar las clases de equivalencia**:
    - Agrupar las cadenas en clases de equivalencia basadas en la relación $\equiv_L$.
3. **Verificar si el número de clases es finito**:
    - Si el número de clases de equivalencia es finito, entonces L es regular.
    - Si el número de clases de equivalencia es infinito, entonces L no es regular.

Un ejemplo aplicando el teorema de Myhill-Nerode. El lenguaje L es equivalente a $\{0^i 1^j : i, j > 0\}$, donde debe haber por almenos un "0" seguidr por un "1", de ahi puede haber una cantidad idefinida de "0" y "1".

- Los cociente de los símbolos seria $\{0, 1\}$
- Le quitamos al lenguaje L cada uno de los símbolos.
- $0^{-1}L = L \cup \{b\}^+ \to L^I$
- $1^{-1}L = \oslash \to L^{II}$
- Ahora hacemos lo mismo para los nuevos lenguajes y asi consecutivamente.
- $0^{-1}L^I = L \cup \{b\}^+ \cup \oslash \to L^I$
- $1^{-1}L^I = \oslash \cup \{b\}^* \to L^{III}$
- $0^{-1}L^{III} = \oslash \to L^{II}$
- $1^{-1}L^{III} = \{b\}^* \to L^{III}$
- Nuestro lenguaje es finito porque tiene una numero finito de clases de equivalencias.

Otro ejemplo aplicando el teorema de Myhill-Nerode pero en donde el lenguaje no es regular. El lenguaje L es equivalente a $\{c^n d^n : n \geq 0\}$, donde tiene que haber la misma cantidad de c y de d.

- Primero hay que encontrar una secuencia infinita que genere infinitos cocientes diferentes en el lenguaje L.
- Primero definimos otro lenguaje A que esta formado por secuencias de c, que tena minimo una c.
- El primer elemento de este lenguaje A se le resta al lenguaje, formando una cadena solo compuesta de d del mismo tamaño del elemento del lenguaje A, y esta misma cadena se le resta al lenguaje.
- Repetimos este proceso con el siguiente elemento del lenguaje A.
- Terminaremos en un proceso que nunca se acabara, por ende podemos determinar que este lenguaje no es regular.
- La forma que podemos formalizar todo esto es de la siguiente forma:

$$\forall p, q \in N, p \neq q$$

$$c^p \text{ no pertence a la misma clase de equivalencia que } c^q$$

$$\downarrow$$

$$d^p \in (c^p)^{-1}L \wedge d^p \notin (c^q)^{-1}L$$

7 Minimización de Estados de un Autómata

La reduction de estados de un autómata es un proceso por el cual se tiene un AF y se reducen la cantidad de estados que tenía originalmente de forma que siga funcionando de la misma forma.
Antes de reducir un automata hay que comprobar si cumple con ciertos requisitos:

- El autómata debe tener un camino viable para todos los estados. En el caso de que haya estados inalcanzables se pueden eliminar.
- El autómata tiene que tener una transición para cada símbolo dentro de cada uno de sus estados.
  - En el caso de que el autómata no este completo se debe agregar las transiciones faltantes.
  - En los estados que no tengan las transiciones para todos los símbolos, añadirles una nueva transición con los símbolos faltantes a un estado nuevo.
  - Este estado nuevo llamado estado sumidero se buclea sobre sí mismo y no es final.

Algoritmo de reducción de estados

1. Condiciones que se tienen que seguir durante cada iteración
   - El número de clases sera igual o mayor que en la iteración anterior.
   - Estados que en un momento determinado del algoritmo pertenezcan a clases diferentes, nunca podrán volver a pertenecer a la misma clase.
2. Primera partición
   - La primera clase solo va estar conformada de los estados no finales.
   - La segunda clase va estar compuesto de estados finales.
3. Segunda pertición
   - Se elabora primera una tabla
     - En las filas se ponen todos los estados del autómata
     - En las columnas todos los símbolos con que trabaja el autómata
     - Por cada fila de la tabla se va a poner la clase a la que pertenece el estado que lleva el símbolo.
   - Una vez llena la tabla se identifican que estados se comportan de manera diferentes y cuales iguales.
   - Se le asigna una clase a todos los estados que se comparten diferentes y se agrupan en una misma clase todos los que tienen un comportamiento similar.

4. Tercera partición

    - Volvemos a hacer otra tabla donde las filas son los estados y las columnas los símbolos.
    - Solo se van a llenar las filas de los estados que comparten comportamiento con otros estados el resto no nos importan.
    - En las filas que nos importan se van a llenar sus columnas con la clase a la que pertenece el estado que lleva el símbolo de esa columna.
    - Una vez llena la tabla se identifican que estados se comportan de manera diferentes y cuales iguales.
    - Seguir haciendo estas particiones hasta que la partición obtenida sea igual a la anterior.

5. Tabla de transiciones final

    - Se va a usar la partición recientemente obtenida.
    - Se crea una tabla donde las filas son todas las clases de la partición y las columnas son los símbolos que se trabajan en el autómata.
    - Para cada fila (o clase) de la tabla se va a poner la clase a la cual pertenece el estado (o estados) el que lleva la transición de ese mismo símbolo.
    - Si la clase tiene múltiples estados no importa mucho ya que sus transacciones llevan a los estados que se encuentran en la misma clase.

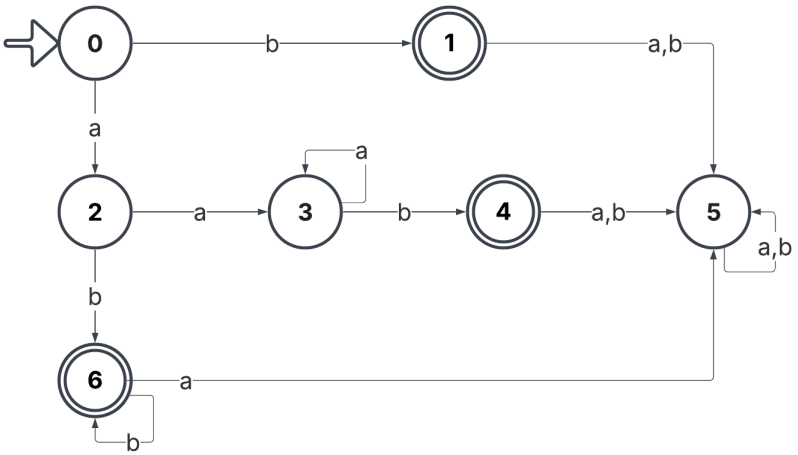
6. Generar el nuevo autómata

    - Con la tabla que se creo anteriormente se puede hacer el autómata reducido.
    - Las clases se vuelven los estados y sus transiciones estan determinadas en la tabla.
    - El estado inicial se determina por la clase que tiene el estado inicial del autómata original.
    - Los estados finales seran los en que tiene por al menos un estado final del autómata original en su clase (Normalmente todo el conjunto de estados estara conformado solo por estados finales).

Ahora un ejemplo de como se veria aplicado la minimización de estados en un autómata. Tenemos nuestro autonoma finito determinista $(Q, \Sigma, \delta, q0, F)$, el cual se encuentra definida como:

- $Q = \{q0, q1, q2, q3, q4, q5, q6\}$
- $\Sigma = \{a, b\}$

- $q0 = q0$
- $F = \{q1, q4, q6\}$
- $\delta = \{(q0, a) = q2(q0, b) = q1, (q1, a) = q5, (q1, b) = q5, (q2, a) = q3, (q2, b) = q6, (q3, a) = q3, (q3, b) = q4, (q4, a) = q5, (q4, b) = q5, (q5, a) = q5, (q5, b) = q5, (q6, a) = q5, (q6, b) = q6\}$



El autómata es completo y no hay estados inaccesibles, entonces no hay modificaciónes

Primera partición Π_1

$$c_1 = \{q0, q2, q3, q5\}$$

$$c_2 = \{q1, q4, q6\}$$

Segunda partición Π_2

	a	b
q0	c1	c2
q1	c1	c1
q2	c1	c2
q3	c1	c2
q4	c1	c1
q5	c1	c1
q6	c1	c2

$$c_1 = \{q0, q2, q3, q6\}$$

$$c_2 = \{q1, q4, q5\}$$

Tercera iteración Π_3

	a	b
q0	c1	c2
q1	c2	c2
q2	c1	c1
q3	c1	c2
q4	c2	c2
q5	c2	c2
q6	c2	c1

$$c_1 = \{q2\}$$

$$c_2 = \{q6\}$$

$$c_3 = \{q0, q3\}$$

$$c_4 = \{q1, q4, q5\}$$

Cuarta iteración Π_4

	a	b
q0	c1	c4
q1	c4	c4
q2	-	-
q3	c3	c4
q4	c4	c4
q5	c4	c4
q6	-	-

$$c_1 = \{q0\}$$

$$c_2 = \{q2\}$$

$$c_3 = \{q3\}$$

$$c_4 = \{q6\}$$

$$c_5 = \{q1, q4, q5\}$$

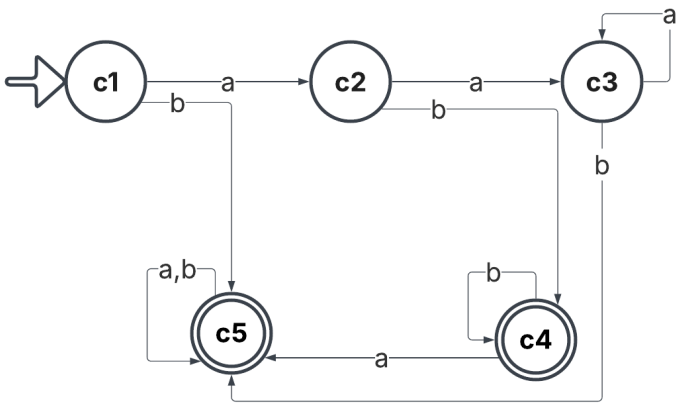
Quinta iteración Π_5

	a	b
q0	-	-
q1	c5	c5
q2	-	-
q3	-	-
q4	c5	c5
q5	c5	c5
q6	-	-

$$\Pi_5 = \Pi_4$$

Se hace la tabla de transiciones final

	a	b
c1	c2	c5
c2	c3	c4
c3	c3	c5
c4	c5	c4
c5	c5	c5



8 Preguntas

1. ¿Qué es un autómata finito?

   - Un autómata finito es un modelo matemático que consiste en un conjunto de estados, un alfabeto de entrada, una función de transición, un estado inicial y un conjunto de estados de aceptación. Procesa una cadena de símbolos y decide si la cadena es aceptada o rechazada basándose en las transiciones entre estados.

2. ¿Cuales son las diferencias entre un AFD y un AFN?

   - En un AFD, para cada estado y símbolo de entrada, hay exactamente una transición definida. En un AFN, para un estado y símbolo de entrada, puede haber cero, una o múltiples transiciones posibles.

3. ¿Qué es una transición épsilon/vacía en un AFN?

   - Una transición épsilon es una transición que ocurre sin consumir ningún símbolo de entrada. Permite que el autómata cambie de estado sin avanzar en la cadena de entrada, lo que añade flexibilidad al diseño del autómata.

4. ¿Qué es la minimización de un autómata finito?

   - La minimización de un autómata finito es el proceso de reducir el número de estados de un AFD sin alterar el lenguaje que reconoce. El objetivo es obtener un autómata equivalente con el menor número posible de estados.

5. ¿Qué es el lenguaje regular y cómo se relaciona con los autómatas finitos?

   - Un lenguaje regular es un tipo de lenguaje formal que puede ser reconocido por un autómata finito. Estos lenguajes pueden describirse mediante expresiones regulares y son la clase más simple de lenguajes en la jerarquía de Chomsky.

9 Bibliografías

- Hopcroft, J.E (2002). *Introducción a la Teoria de Autómatas, Lenguajes y Computación.* Madrid: Pearson Education.
- Kelley, D. (1995). *Teoría de Autómatas y Lenguajes Formales.* Madrid: Pretince Hall.
- Muñoz, R. C. (2002). *Lenguajes, gramáticas y autómatas. Curso básico.* Ciudad de México: Alfaomega