CAPÍTULO 6: Autómatas reconocedores de Lenguajes

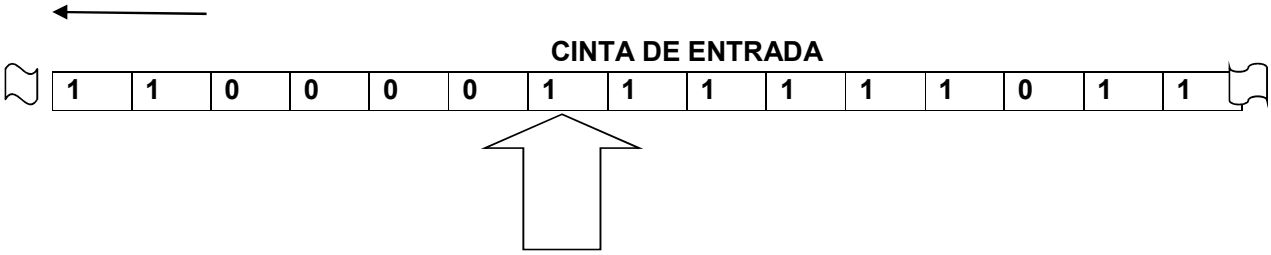
Existen distintos modelos matemáticos, maquinas u autómatas virtuales capaces de reconocer las sentencias o palabras de los distintos lenguajes. De acuerdo de que gramática deriven sus palabras Según la clasificación de Noam Chomsky éstas maquinas podrán ser:

1. Lenguajes que derivan de gramáticas Tipo 0, gramáticas sin restricciones (GSR), presentaremos como reconocedora de sus palabras al modelo de Turing conocida como Máquina de Turing (MT)
2. Lenguajes que derivan de gramáticas Tipo 1, gramáticas sensibles o dependientes del contexto (GSC o GDC), el autómata reconocedor de estos lenguajes serán Autómatas Linealmente Acotado (ALA)
3. Lenguajes que derivan de gramáticas Tipo 2, gramáticas independientes del contexto (GIC), el autómata reconocedor de estos lenguajes serán los Autómatas de Pila Push Down (APPD)
4. Lenguajes que derivan de gramáticas Tipo 3, gramáticas regulares (GR), el autómata reconocedor de estos lenguajes serán Autómatas Finitos (AF).

AUTOMATAS FINITOS

Los autómatas finitos reconocen se pueden representar gráficamente, en forma intuida, como una cinta y un control de estados o cabeza lectora.
La cinta contiene símbolos de un determinado vocabulario y se mueve en un solo sentido.

Dirección que se mueve la cinta, nunca vuelva atrás



La cinta puede ser infinita **Cabezal**

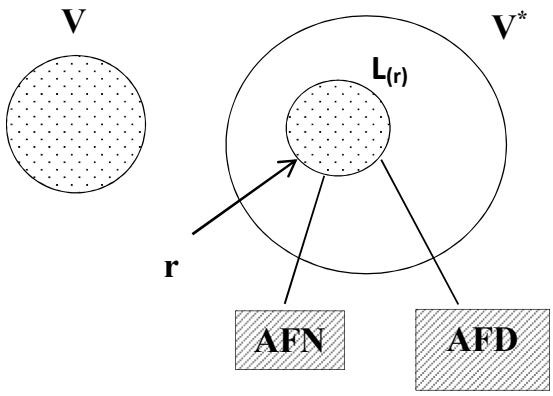
La sentencia o palabra colocada en la cinta será leída por el AF y aceptada si el cabezal llega a un estado final.
Podemos decir entonces que un AF es una máquina “reconocedora”, ya que a partir de una cadena de entrada, solo dice “si” o “no”. Lo cual indica si esa cadena pertenece o no al lenguaje reconocido o aceptado por la máquina. A partir de una gramática se puede construir una máquina reconocedora o aceptadora del lenguaje generado por esa gramática, de tal forma que cuando reciba a su entrada una determinada cadena de símbolos, el autómata indicará si dicha cadena pertenece o no al lenguaje. Una máquina reconoce un lenguaje L si es capaz de reconocer todas las sentencias pertenecientes a L y de no reconocer ninguna sentencia que no pertenezca a L. Los lenguajes regulares pueden describirse por medio de un AF. Un autómata es un modelo matemático, me da el comportamiento y esto se puede simular en computadora (programas de simulación).

Un autómata finito puede representar un reconocedor de un lenguaje. Es un programa que toma como entrada una cadena x y responde "SI", si x es una frase del programa y "NO", en caso contrario. El autómata finito representa una expresión regular y reconoce gramáticas de tipo 3.

Un AF tiene un conjunto de estados y su control se mueve de estado en estado, en respuesta a entradas externas. Estas entradas forman las cadenas a ser analizadas. Los estados de un AF, son de tres tipos: estado inicial, que permite empezar la ejecución del autómata; estados finales o estados "de aceptación" que permiten realizar la salida de aceptación de la cadena de entrada en el caso de que no haya más símbolos en la entrada, y estados intermedios, que son los que permiten pasar del estado inicial a algún estado final.

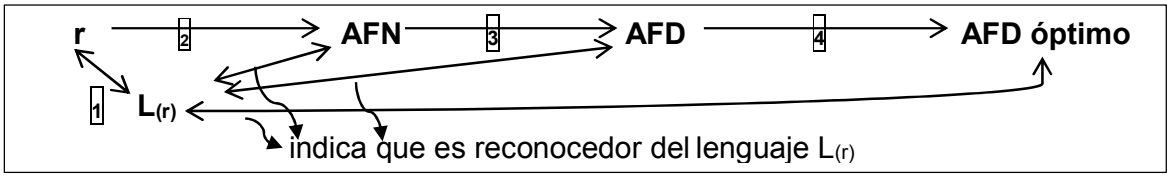
Los AF se dividen en diversas clases, dependiendo de si su control es "determinista" **A**utómata **F**inito **D**eterminista – **AFD** (desde un estado para cada símbolo se puede producir una transición a solo un estado, lo que significa que el autómata no puede estar en más de un estado simultáneamente) o "no determinista" **A**utómata **F**inito **N**o-Determinista – **AFN** (un estado puede tener más de una transición para el mismo símbolo de entrada, lo que significa que el autómata puede estar en varios estados al mismo tiempo).

Para un mismo lenguaje regular puedo tener un reconocedor que sea AFN y otro que sea AFD.



El AFD por lo general presenta más estados que el AFN, o sea, es más grande ⇒ ocupa más lugar (esto es una desventaja).

El AFD llega a la respuesta con menor cantidad de operaciones ⇒ es más rápido (es una ventaja)



1 →Expresión regular **r** que representa el lenguaje **L(r)**
2 →Método por el cual a partir de una expresión regular dada se puede obtener un AFN
3 →Proceso para construir un reconocedor de un lenguaje regular
4 →Método para construir un AFD óptimo

4.1. AUTÓMATAS FINITOS NO DETERMINISTAS

Un autómata finito "no determinista" AFN tiene la capacidad de estar en varios estados simultáneamente. El término "no determinista" hace referencia al hecho de que para cada entrada devuelve un conjunto de cero, uno o más estados.

Definición formal: Un AFN es un modelo matemático formado por:

1) Un conjunto finito de "estados" S.
2) Un alfabeto de "símbolos de entrada" V
3) Un estado S_0 que se considera "de inicio" o "inicial" ($S_0 \in S$)
4) Un conjunto de estados F considerados "estados de aceptación" o "finales". ($F \subseteq S$)
5) Una función de transición **mueve** que transforma pares estado-símbolo en conjuntos de estados. "**mueve**" es una función que toma como argumentos un estado de S y un símbolo de entrada de V y devuelve un subconjunto de S.

mueve : S x V → P(S) conjunto de estados

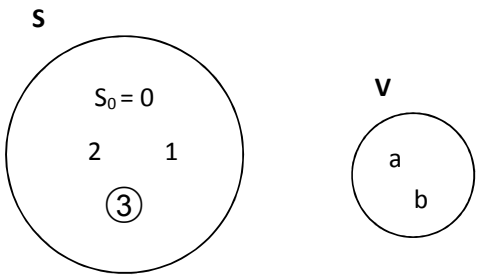
Es decir, un AFN se puede denotar con la quíntupla : AFN = {S, V, S_0, mueve, F}

4.2. TABLA DE TRANSICIONES

La manera más sencilla de implementar esto en un computador es por medio de un cuadro o matriz en el cual a cada estado le corresponde una fila y a cada símbolo una columna (tabla de doble entrada). Esta tabla recibe el nombre de "Tabla de Transiciones" del autómata.

Cada **entrada** representa el conjunto de estados que puede ser alcanzado por una transición desde el estado i con la entrada j.

Ejemplo: el siguiente autómata es un AFN basado en el lenguaje representado por la expresión regular (a | b)*abb. La tabla de transiciones que representa este AFN sería:



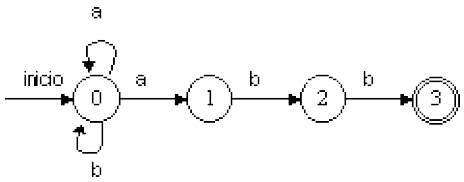
Símbolos / Estados	a	b
0	{0, 1}	{0}
1	∅	{2}
2	∅	{3}
3	∅	∅

Nótese que la tabla de transiciones sirve para especificar la función de transición del AFN, y que cada anotación en la tabla es un conjunto, aunque dicho conjunto tenga un solo elemento. También debe señalarse que, si no existe ninguna transición a partir de un estado para un símbolo determinado, el valor correcto que hay que colocar en la tabla es, el conjunto vacío.

4.3. GRAFO DE TRANSICIONES

Un AFN también se puede representar mediante un grafo dirigido y etiquetado llamado "**grafo de transiciones",** en el que los nodos representan a los estados del autómata y las aristas etiquetadas y orientadas representan la función mueve, es decir, las transiciones. Las aristas pueden etiquetarse con el símbolo especial ε.

El siguiente es el grafo de transiciones que reconoce el lenguaje (a | b) * a b b



Estados: 0, 1, 2, 3 Estado inicial: 0

Alfabeto: { a, b } Estados finales: { 3 }

ACEPTACION DE UNA CADENA POR PARTE DE UN AFN

Se dice que un AFN **acepta** una cadena de entrada x si hay algún "camino" en el grafo de transiciones desde el estado de inicio a algún estado de aceptación, tal que concatenando las etiquetas de las sucesivas aristas a lo largo del "camino" se obtiene x (la cadena x).

"movimiento" → pasaje por una arista
"camino" → una sucesión de movimientos

Ej:

Dada la cadena x = aba un "camino" sería: $0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{a} 0$

La cadena fue leída completa y el estado de llegada es el estado 0, el cual no es de aceptación. Intentamos otro análisis:

$$0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{a} 1$$

Nuevamente termina en no aceptación, probamos otro análisis: $0 \xrightarrow{a} 1 \xrightarrow{b} 2$ #

No puedo avanzar con una a, ∴ el proceso aborta. La lectura de la cadena no fue competa. No hay más pruebas posibles. Hicimos todos los intentos y en ninguno quedó en el estado de aceptación, ∴ esta cadena no es aceptada.

x = aabb → algunos "caminos" serían:

$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{b} 0$ no salió del S_0

$0 \xrightarrow{a} 1 \xrightarrow{a}$ # aborta

$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$ ✓ encontramos un camino en el grafo, ∴ la cadena x = aabb es aceptada.

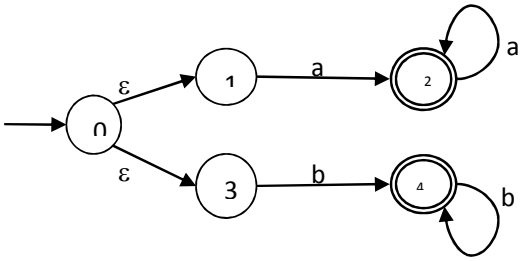
Concluimos que:

En presencia de un AF y de una cadena hay que probar todas las posibilidades, si ninguna llega a aceptación entonces la cadena no es aceptada, si al menos un camino tiene éxito la cadena es aceptada. Puede haber más de una secuencia de movimientos, "camino", que conduzca a un estado de aceptación.

El lenguaje del grafo anterior sería {a; b}* {a} {b} {b} y la expresión regular correspondiente sería:

(a | b)* abb

Otro ej. de AFN:



La arista $\varepsilon \Rightarrow$ no símbolo (ningún símbolo)

Siempre puedo suponer que un símbolo de la cadena está precedido por $\varepsilon \Rightarrow x = a \cong \varepsilon\, a$

$0 \xrightarrow{\varepsilon} 1 \xrightarrow{a} 2$ ✓

$0 \xrightarrow{\varepsilon} 3$ # aborta

para $x = a = \varepsilon\, a$

Aceptadas: a, aa, aaa, ……, b, bb, bbb …….

∴ el lenguaje sería: a.a* | b.b*, o sea: $\{a\}^+ \cup \{b\}^+$ y la expresión regular $a^+ \mid b^+$ (no contiene la vacía)

El AFN anterior acepta las cadenas abb, aabb, babb, aaabb.

El **lenguaje definido por un AFN** es el conjunto de cadenas que acepta.

4.4. AUTÓMATAS FINITOS DETERMINISTAS

Un AFD es un caso especial de un AFN en el cual:

1) ∄ transiciones ε
2) Para cada estado S y cada símbolo de entrada a, hay a lo sumo una arista etiquetada con a saliente del estado S

Un autómata finito "determinista" (AFD) siempre está en un solo estado después de leer cualquier secuencia de entrada. El término "determinista" hace referencia al hecho de que, para cada entrada, existe un único estado al que el autómata pueda llegar partiendo del estado actual. Por lo tanto es fácil determinar si un AFD acepta una cadena, ya que hay a lo sumo un camino desde el estado de inicio etiquetado con esa cadena.

Definición formal: Un AFD es un modelo matemático formado por:

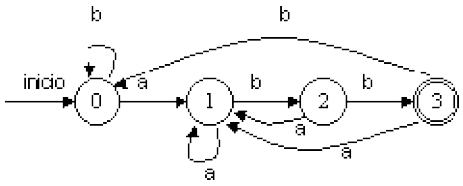
1) Un conjunto finito de "estados" S.
2) Un alfabeto de "símbolos de entrada" V
3) Un estado S_0 que se considera "de inicio" o "inicial" ($S_0 \in S$)
4) Un conjunto de estados F considerados "estados de aceptación" o "finales". ($F \subseteq S$)
5) Una función de transición "**mueve**" que toma como argumentos un estado de S y un símbolo de entrada de V y devuelve un estado de S.

mueve : $S \times V \rightarrow P(S)$ conjunto de estados

Es decir, un AFD se puede denotar con la quíntupla : $A = \{S, V, S_0, \text{mueve}, F\}$

Observación: en una AFD cualquiera sea el estado y cualquiera sea el símbolo no habrá arista saliente con ese símbolo o habrá a lo sumo 1, o sea, para cada estado y para cada símbolo existe una única alternativa.

El siguiente grafo de transiciones reconoce el mismo lenguaje que el AFN dado como 1er. ejemplo ((a | b)* abb) pero respetando un AFD.



La cadena en estudio se llama “cadena candidata”. x = bababbab → termina en ② ⇒ no aceptada

TABLA DE TRANSICION PARA EL AFD

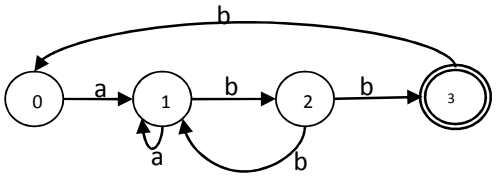
Símbolos / Estados	a	b
0	1	0
1	1	2
2	1	③
③	1	0

Las imágenes son conjuntos, pero al tratarse de un AFD para cada par estado–símbolo el conjunto es vacío (**-**) o tiene un único estado

Este autómata tiene la particularidad de que para cualquier estado – símbolo hay una llegada, no tiene vacíos (**-**), se llama “tabla llena”. Desde el punto de vista práctico esto quiere decir que cualquiera sea la cadena, ya sea aceptada o no, va a ser leída completa, no aborta.

Ej.: “Tabla no llena”

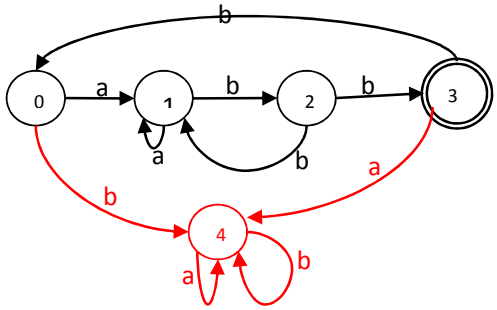
Símbolos / Estados	a	b
0	1	-
1	1	2
2	1	③
③	-	0



Simulación de un AFD en una computadora (Aho pag.119)

En un archivo tenemos una cadena de entrada **x**, no sabemos cuántos símbolos tiene, por lo tanto hay que indicar el fin de la cadena con un carácter especial llamado **eof** (end-of-file). En la memoria tendremos una variable **S** para el estado y una variable **c** para símbolo. Por otro lado necesitamos tener disponible la **tabla de transiciones** (matriz). Una hipótesis simplificativa es que la tabla está llena (tabla completa). Hay un artificio para completar una tabla con vacíos (**-**) sin modificar las propiedades de la tabla, consiste en agregar un estado adicional para rellenar los vacíos (**-**)

Símbolos / Estados	a	b
0	1	**4**
1	1	2
2	1	③
③	**4**	0
4	**4**	**4**



Una función: sgtecar( ) que toma el siguiente carácter de la entrada (la 1ª vez el 1º), es decir, captura el carácter que sigue en la secuencia.

Y una función: mueve(S, c) que captura de la matriz el estado que le corresponde al estado S y al símbolo c.

Algoritmo para leer una cantidad desconocida de datos:

Leer el 1er. dato
Mientras dato leído ≠ centinela
Procesar el dato leído
Leer el sgte. dato

Hay que implementar algún mecanismo para individualizar los estados de aceptación (los estados de F), porque cuando llegue al final del proceso debo determinar si el estado en que quedó ∈ o ∉ a F.

Arranco en S_0 (estado inicial) y leo el 1er. Símbolo, con mueve saco el nuevo estado y lo asigno a S, leo el sgte. carácter (símbolo) y así sucesivamente, cuando símbolo leído = eof, el estado que quedó en S es el final.

```
S ← S0;
c ← sgtecar( );
while (c ≠ eof)
    S ← mueve(S, c);
    c ← sgtecar( );
if (S está en F)
    retornar ( 1 );
else
    retornar ( 0 );
```

4.5. PASO DE UNA EXPRESIÓN REGULAR A UN AFN (r → AFN)

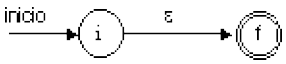
Para el pasaje de una expresión regular a un AFN se utiliza un método llamado **Construcción o Método de Thompson (r → AFN)**, este método tiene 3 reglas y tiene carácter recursivo.

La e.r. tiene una estructura en la que aparecen operandos (ε y símbolos del alfabeto) y operadores (* **.** |). Primero se hace un análisis sintáctico de la expresión regular en sus subexpresiones correspondientes que la constituyen. Se construye un AFN para cada uno de los símbolos básicos de la expresión regular r (ε y símbolos del alfabeto).

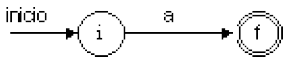
Luego, guiándose por la estructura sintáctica de r, se combinan inductivamente los AFN hasta obtener el AFN de la expresión completa.

La construcción de los AFN sigue las siguientes reglas:

1) Para cada ocurrencia de ε en r, construir el AFN que reconoce el lenguaje { ε }

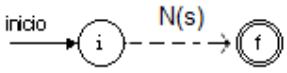


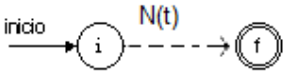
2) Si a en V, por cada ocurrencia de a en r construir el AFN que reconoce { a }



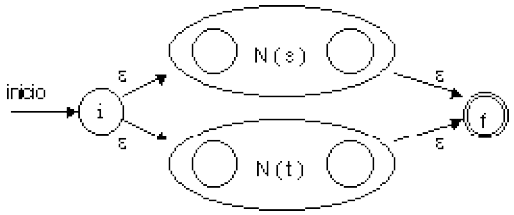
3) Si N (s) y N (t) son los AFN correspondientes a las e.r. s y t, entonces:

3.a) Para la e.r. s | t, construir el AFN que reconozca la unión de los dos lenguajes N (s | t) (todas las cadenas de los 2 lenguajes)





Se colocan en paralelo los 2 AFN y se conectan con ε. Los estados de inicio y aceptacion de cada uno dejan de serlo; hay una transición ε desde el nuevo **i** a los estados de inicio de $N_{(s)}$ y $N_{(t)}$ y hay una transicion ε desde los estados de aceptación de $N_{(s)}$ y $N_{(t)}$ al nuevo estado de aceptacion **f**. observe que todos los caminos desde **i** a **f** deben pasar exclusivamente por $N_{(s)}$ o por $N_{(t)}$.

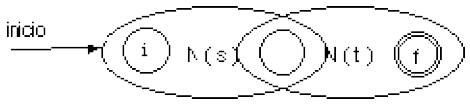


Reconoce $L_{(s)}$ **U** $L_{(t)}$

3. b) Para la e.r. s.t, construir el AFN compuesto N (s . t)

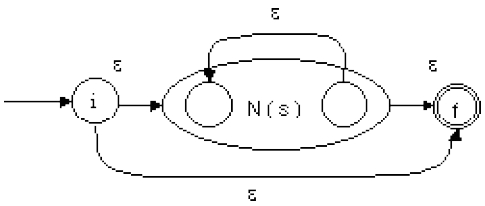
Se colocan en serie los dos AFN y el estado de inicio de $N_{(s)}$ se convierte en el estdo de inicio de la AFN compuesta. El estado de aceptación de $N_{(t)}$ se convierte en el estado de aceptación de la AFN compuesta. El estado de aceptación de $N_{(s)}$ se fusiona con el estado de inicio de $N_{(t)}$.

Un camino desde i a f debe pasar primero a través de $N_{(s)}$ y después por $N_{(t)}$.



Reconoce $\mathbf{L_{(s)} \bullet L_{(t)}}$

3.c) Para la e.r. s *, construir el AFN compuesto N (s) *



Reconoce $\mathbf{(L_{(s)})^*}$

i es un nuevo estado de inicio y f un nuevo estado aceptación. Se puede ir desde i a f directamente, a lo largo de una arista etiquetada con ε ($\varepsilon \in (L_{(s)})^*$), o se puede ir desde i a f por $N_{(s)}$ una o más veces.

3.d) Para la expresión regular (s), construir el AFN N (s)

Resumen del Método de Thompson

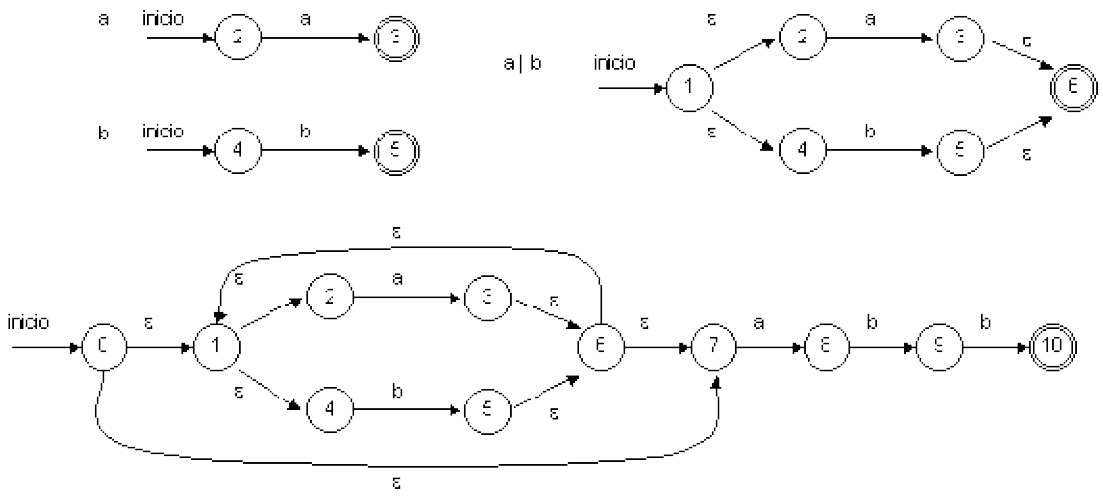
Primero se hace un análisis sintáctico de **r** en sus subexpresiones constituyentes. Después se construyen los AFN para cada uno de los símbolos básicos de **r**. es importante comprender que si un símbolo **a** aparece varias veces en **r**, se construye un AFN independiente en cada caso (reglas 1 y 2). Después, guiándose por la estructura sintáctica de **r**, se combinan inductivamente esos AFN, utilizando la regla 3 hasta obtener el AFN para la expresión completa. Cada vez que se construye un nuevo estado se le debe dar un nombre distinto.

Propiedades del AFN $N_{(r)}$ resultante:

- ✓ El AFN generado tiene a lo sumo el doble de estados que de símbolos y operadores en r, ya que en cada paso de la construcción se crean a lo sumo 2 nuevos estados.
- ✓ El AFN N (r) tiene exactamente un estado de inicio y otro de aceptación (f no tiene transiciones salientes).
- ✓ Cada estado del AFN N(r) tiene una transición saliente con un símbolo en V o a lo sumo 2 transiciones ε, salientes.

Ej:

r = (a | b) * a b b



4.6. CONVERSIÓN DE UN AFN EN UN AFD (AFN → AFD)

El **objetivo** de esta conversión consiste en construir un AFD que reconozca el mismo lenguaje que un AFN del cual se parte. Esto se logra mediante un algoritmo llamado de **construcción de subconjuntos o método de los subconjuntos**.

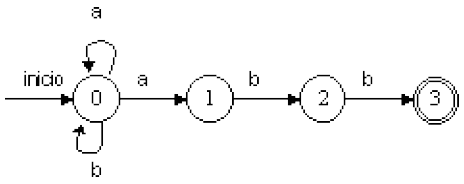
La base del proceso de conversión consiste en relacionar cada estado del AFD con un conjunto de estados del AFN.

En la tabla de transiciones de un AFN, cada entrada es un conjunto de estados, mientras que en la tabla de un AFD, cada entrada es un estado a lo sumo.

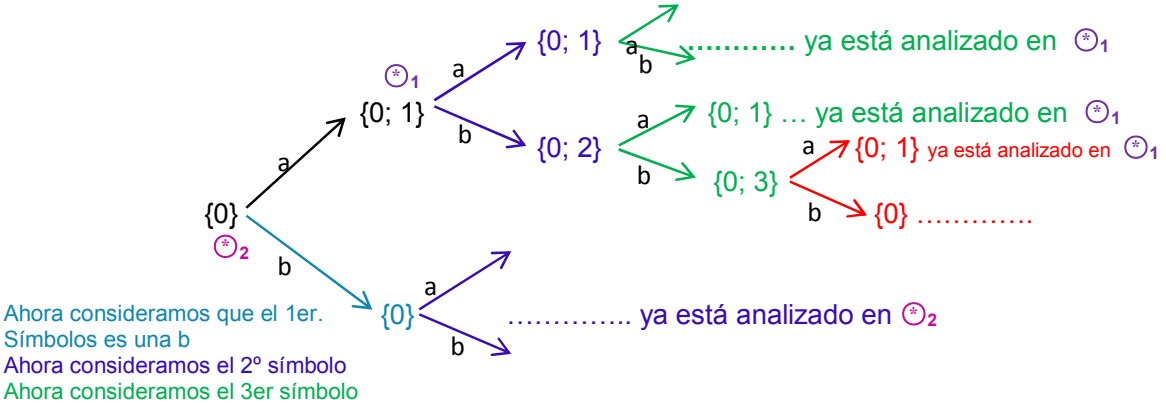
El AFD usa un estado para localizar todos los posibles estados en los que puede estar el AFN después de leer cada símbolo de la entrada.

Llamamos N al AFN de entrada y D al AFD de salida.

Ej: Tenemos el sgte. AFN: (a|b)abb (sencillo pues no tiene transiciones ε)



A partir del mismo vamos a obtener un AFD. Pensemos en el conjunto {0} (conjunto que solo tiene el estado 0). Imaginemos una cadena de entrada que tiene como 1er. Simbolo una a y veamos a donde podemos llegar (obviamos los abortos).



En la tabla de transiciones de un AFN, cada entrada es un conjunto de estados, mientras que en la tabla de un AFD, cada entrada es un estado a lo sumo.

Se relaciona cada estado del AFD con un conjunto de estados del AFN

AFN

Est. \ Símb.	a	b
A {0}	{0; 1}	{0}
B {0; 1}	{0; 1}	{0; 2}
C {0; 2}	{0; 1}	{0; 3}
D {0; 3}	{0; 1}	{0}

AFD

Estados \ Símbolos	a	b
A	B	A
B	B	C
C	B	Ⓓ
Ⓓ	B	A

Ⓓ → es el estado de aceptación porque contiene el estado ③

Para todo par est-simb hay a lo sumo 1 estado de llegada

Después de leer la entrada $a_1a_2a_3$…….a_n, el AFD se encuentra en un estado que representa al subconjunto T de los estados del AFN alcanzables desde el estado inicial del AFN por medio de algún camino etiquetado con $a_1a_2a_3$…….a_n.

El AFD simula "en paralelo" todos los posibles caminos que la AFN puede recorrer con una determinada cadena de entrada.

Ahora veremos el método general (con transiciones ε):

Imaginemos que:

S: estado del AFN

T: conjunto de estados del AFN

a: símbolo de entrada

entonces definimos:

El algoritmo construye una tabla de transiciones para D teniendo en cuenta las siguientes **operaciones**, para localizar los conjuntos de estados del AFN (donde S representa un estado del AFN y T, un conjunto de estados del AFN).

cerradura-ε (s) → Conjunto de estados del AFN alcanzables desde el estado S del AFN con transiciones ε solamente.

cerradura-ε (T) → Conjunto de estados del AFN alcanzables desde algún estado S en T con transiciones ε solamente. Incluye los propios estados de T, todo estado se alcanza a si mismo por ε

mueve (T, a) → Conjunto de estados del AFN alcanzables con una única transición a desde algún estado S de T.

El primer paso consiste en detectar el estado de inicio de N, llamado s_0, que definirá cerradura-ε (s_0).

Este es un conjunto de estados alcanzables desde s_0.

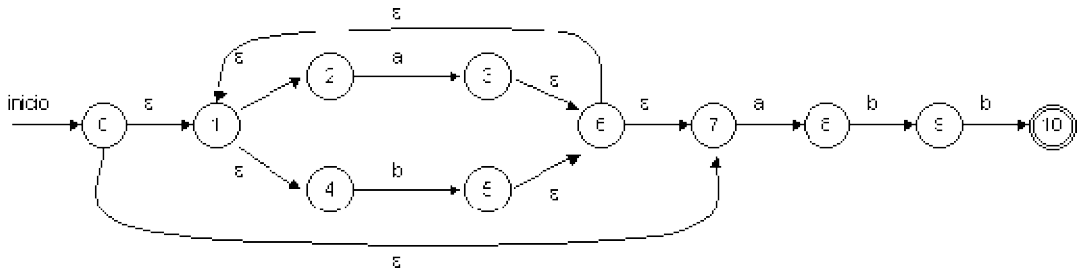
Ahora corresponde tomar cada símbolo de entrada a y ejecutar la operación mueve (T, a), para obtener cada nuevo conjunto de estados, que se definirá por cerradura-ε (mueve (T, a)).

Luego se debe ingresar el conjunto obtenido en cada caso como nuevo miembro de la tabla de transiciones D.

Para cada conjunto obtenido, que no haya sido analizado se ejecuta nuevamente la operación mueve (T, a), hasta que se hayan recorrido todos.

Finalmente se definen como estados de aceptación de D a aquellos conjuntos de estados del AFN que contengan al menos un estado de aceptación de N.

Ej: r = (a | b) * a b b



cerradura-ε (0) = { 0, 1, 2, 4, 7 } = A

mueve (A, a) = { 3, 8 }

cerradura-ε ({ 3, 8 }) = { 1, 2, 3, 4, 6, 7, 8 } = B

mueve (A, b) = { 5 }

cerradura-ε ({ 5 }) = { 1, 2, 4, 5, 6, 7 } = C

mueve (B, a) = { 3, 8 }

cerradura-ε ({ 3, 8 }) = { 1, 2, 3, 4, 6, 7, 8 } = B

mueve (B, b) = { 5, 9 }

cerradura-ε ({ 5, 9 }) = { 1, 2, 4, 5, 6, 7, 9 } = D

mueve (C, a) = { 3, 8 }

cerradura-ε ({ 3, 8 }) = B

mueve (C, b) = { 5 }

cerradura-ε ({ 5 }) = C

mueve (D, a) = { 3, 8 }

cerradura-ε ({ 3, 8 }) = B

mueve (D, b) = { 5, 10 }

cerradura-ε ({ 5, 10 }) = { 1, 2, 4, 5, 6, 7, 10 } = E

mueve (E, a) = { 3, 8 }

cerradura-ε ({ 3, 8 }) = B

mueve (E, b) = { 5 }

cerradura-ε ({ 5 }) = C

Los 5 conjuntos de estados construídos son:

A = { 0, 1, 2, 4, 7 }

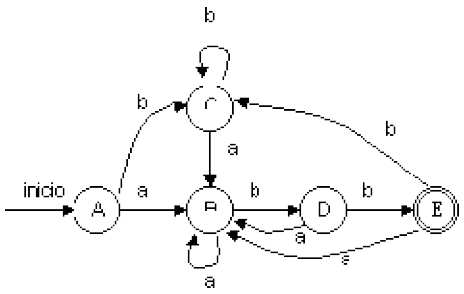
B = { 1, 2, 3, 4, 6, 7, 8 }

C = { 1, 2, 4, 5, 6, 7 }

D = { 1, 2, 4, 5, 6, 7, 9 }

Estados \ Símbolos	a	b
A	B	C
B	B	D
C	B	C
D	B	Ⓔ
Ⓔ	B	C

E = { 1, 2, 4, 5, 6, 7, ⑩ } contiene el est. de aceptación inicial ∴ es est. de aceptación



4.7. MINIMIZACIÓN DEL N° DE ESTADOS DE UN AFD

Condición: Tabla Completa ⇒ ninguna cadena aborta, todas son leídas totalmente.

Se debe suponer inicialmente que tenemos un AFD llamado M con un conjunto de estados S y un alfabeto de símbolos de entrada $\sum$.

Se supone también que cada estado tiene una transición con cada símbolo de la entrada. Si no fuera así, se puede introducir un nuevo "**estado inactivo** δ", con transiciones de δ a δ con todas las entradas y añadir una transición desde el estado s al δ en la entrada a si no hubo transición desde s en a.

Se dice que una cadena **w distingue al estado s del estado t** si, empezando con el AFD M en el estado s y alimentándolo con w, se termina en un estado de aceptación, pero comenzando en t, no se termina en un estado de aceptación o viceversa.

Se puede definir un algoritmo que **minimice** el n° de estados del AFD, que funciona encontrando todos los grupos de estados que pueden ser diferenciados por una cadena de entrada. Cada grupo que no puede diferenciarse, se fusiona en un único estado.

Las características de dicho algoritmo son las siguientes:

Entrada: un AFD M con un conjunto de estados S, un alfabeto V, y transiciones para todos los estados y entradas de símbolos, un estado de inicio S_0 y un conjunto de estados de aceptación F.

Salida: un AFD M ' que acepta el mismo lenguaje que M y tiene el menor número de estados posible.

Procedimiento

1) Construir una partición inicial π del conjunto de estados con 2 grupos: los estados de aceptación F y los estados de no-aceptación S-F.

2) Construir una nueva partición π_{nueva} en la que: para cada grupo de π, dividirlo en subgrupos tales que, dos estados s y t del grupo están en el mismo subgrupo si, para todos los símbolos de entrada a, los estados s y t tienen transiciones en a hacia estados del mismo grupo de π.

Finalmente sustituir el grupo de π_{nueva}, por todos los subgrupos.

3) Si $\pi_{nueva} = \pi$ hacer $\pi_{final} = \pi$ e ir al paso 4)

Sino, repetir el paso 2) con $\pi = \pi_{nueva}$.

4) Elegir un estado en cada grupo de π_{final} como representante del grupo. Los representantes serán estados del AFD M ' reducidos.

Se elige el estado inicial de M como representante de estado inicial de M ' y los estados finales de M como representantes de finales de M '.

5) Si M ' tiene un "**estado inactivo**", δ, que no es de aceptación y que tiene transiciones hacia él mismo con todos los símbolos de entrada, eliminarlo de M '.

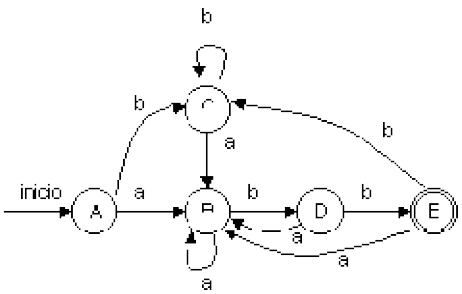
Eliminar también todos los estados que no sean alcanzables desde el estado inicial y todas las transiciones a δ desde otros estados se convierten en indefinidas.

Ahora determinamos como se fija π_0 y como se pasa de π a π_{nueva}

$\pi_0 \longrightarrow$ se separan en partes $\neq$ los estados de aceptación de los de no aceptación (la cadena ε) distingue cualquier estado de aceptación de cualquiera de no aceptación.

$\pi \rightarrow \pi_{nueva} \longrightarrow$ llamamos **G** a cada parte y se toma una parte por vez. Considerando cada parte **G** y cada símbolo **a,** entonces sencillamente se separan en partes $\neq$ de **G** los estados de **G** cuyas transiciones **a** conducen a estados pertenecientes a partes $\neq$ de π. Se mantienen reunidas en una parte de **G** todos los estados de **G** que con cada uno de los símbolos conducen a estados de una misma parte de π.

Ej: Dado el siguiente AFD, que corresponde a la e.r.: (a | b)* .a.b.b



Símbolos / Estados	a	b
A	B	C A
B	B	D
~~C~~	~~B~~	~~C~~
D	B	Ⓔ
Ⓔ	B	C A

Partición inicial π_0:

Grupos:

(E) estado de aceptación

(ABCD) estados de no-aceptación

(E) no se puede dividir, así que va a π_{nueva}

(ABCD) *con la entrada a:* se va en todos los casos a estados del mismo grupo

con la entrada b: A,B,C $\rightarrow$ van al grupo (ABCD)

D $\rightarrow$ va al grupo (D)

Entonces π_1: (ABC) (D) (E)

(ABC) *con la entrada a:* no hay división

con la entrada b: A,C $\rightarrow$ van al grupo (ABC)

B $\rightarrow$ va al grupo (D)

Entonces π_2: (AC) (B) (D) (E)

Ya no se pueden hacer más divisiones entonces:

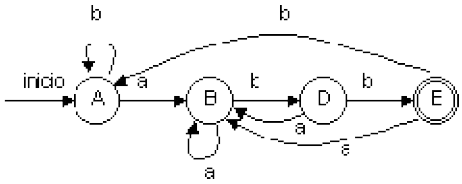
π_{final}: (AC) (B) (D) (E)

De modo que A y C son equivalentes $\Rightarrow$ eliminamos uno de ellos, en este caso C. Se elige A (estado inicial del original) como el representante de (AC) y B, D, E como representantes de los respectivos grupos. El estado eliminado lo tacho de la tabla y lo que llega a ese estado lo reemplazo por su equivalente. A es el estado de inicio y E, el de aceptación

La tabla de transiciones queda definida de la siguiente manera:

Estados \ Símbolos	a	b
A	B	A
B	B	D
D	B	Ⓔ
Ⓔ	B	A

El autómata reducido queda así:



Algoritmo – Minimización del nº de estados de un AFD (Aho pag. 145)

$\pi_0 \leftarrow$ {est-de-aceptación} {est-de-no-aceptación}	
$\pi_{nueva} \leftarrow \pi_0$	
	$\pi \leftarrow \pi_{nueva}$
	Generación de π_{nueva} a partir de π
repetir hasta $\pi_{nueva} = \pi$	
$\pi_{final} \leftarrow \pi_{nueva}$	

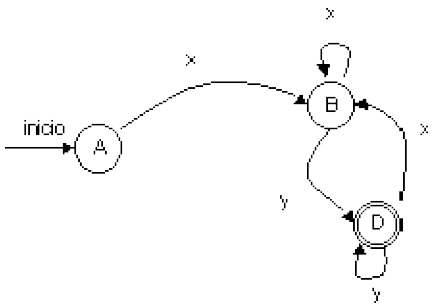
Para c/parte G de π	
	Partición de G en subconjuntos tales que 2 estados s y t de G están en el mismo subconjunto $\Leftrightarrow$ para todo símbolo de entrada a los estados s y t tienen transiciones a hacia estados de la misma parte de π
	Sustitución de G en π_{nueva} por el conjunto de todos los subconjuntos formados

Ej: r = x (x | y) * y

La tabla de transiciones queda definida de la siguiente manera:

Estado	Símbolo de entrada	
	x	y
A	B	-
B	B	D
D	B	D

El autómata reducido queda así:



6.8 AUTOMATA DE PILA PUSH-DOWN (APD)

Reconocedor de palabras de lenguajes que derivan de **gramáticas GIC**. Gráficamente lo podemos imaginar con los siguientes elementos:

1

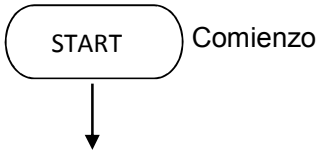
Cinta infinita ⍉ significa celda vacía

x	x	x	y	y	y	⍉	⍉	⍉	⍉

Movimiento de la cinta, una vez leída una celda nunca vuelva atrás

xxxyyy: cadena cargada en la cinta, el resto tiene blancos

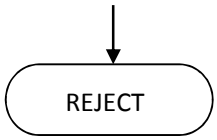
2



Comienzo

3 Estados de detención

stado de aceptación



Estado de NO aceptación

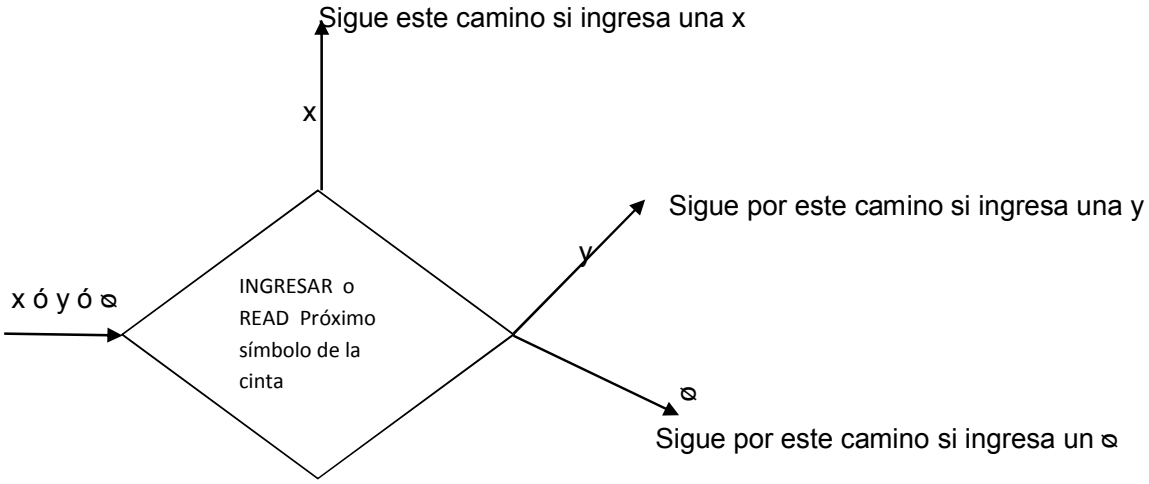
En ambos estados de DETENCION, se puede entrar pero no se puede salir de ellos.

4 Estructura de almacenamiento Pila o Stack LIFO (last in first out, último en entrar primero en salir)

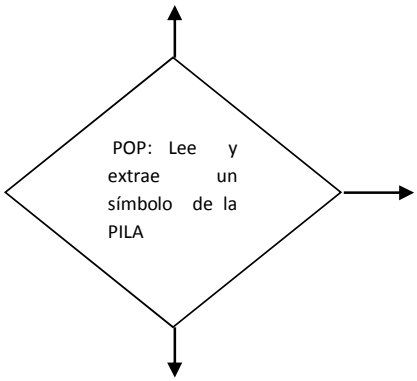
Stack o PILA: vacía

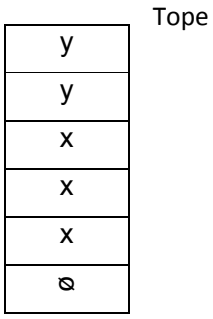
⍉	TOPE
⍉	
⍉	
⍉	
⍉	
⍉	

5 Read, Leer o Ingresar de la cinta

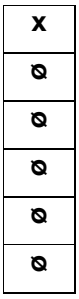


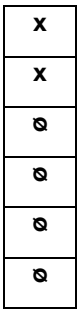
Acción POP: lee un elemento de la pila y lo saca de la misma , comenzando desde el TOPE. Para llegar a Leer el símbolo x de la PILA debe leer **y** dos veces (POP y, POP y, llegando a x), **y** no estará más en la PILA.

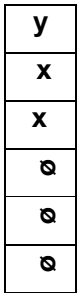




6 Acción PUSH Coloca un elemento en la pila comenzando desde el tope, la pila al principio esta vacía







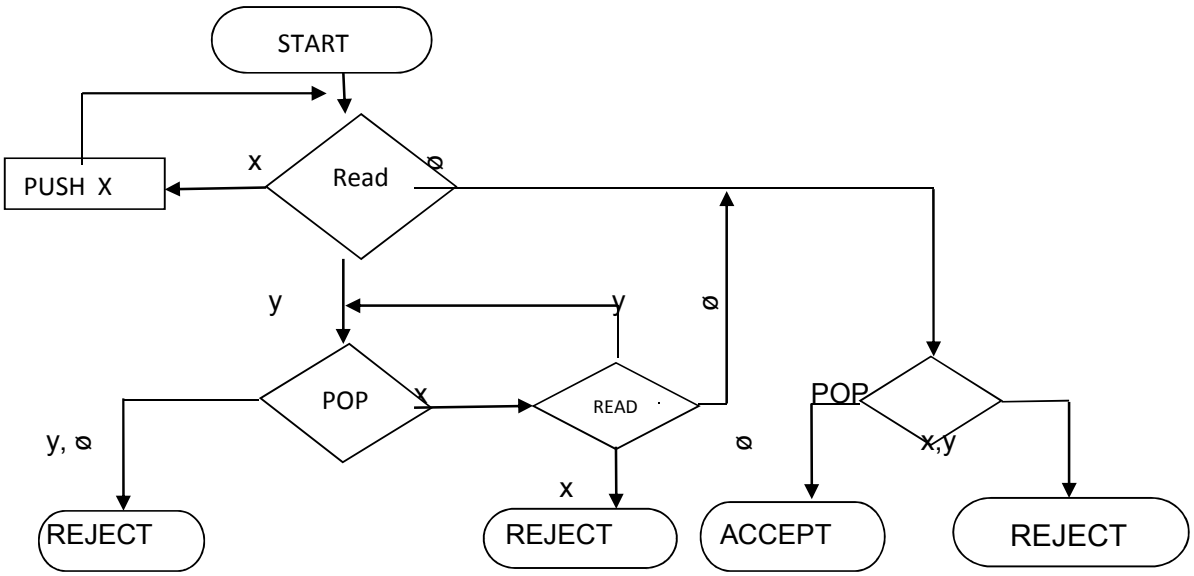
7 Un alfabeto de entrada de la cinta ALFC, que no tiene el carácter ⍉ de espacio en blanco

8 Un alfabeto de la PILA ALFP

APDD reconocedor de palabras de una gramática GIC

Un **APD** es **determinista (APDD)** si para toda cadena de entrada tiene un único camino a través de la máquina.

Un **APD** es **NO determinista (APND)** existen posibles caminos electivos a través de la máquina



Recorrido

Suponemos que en la cinta tenemos los símbolos **xxxyyy**
Este APD funciona de la siguiente manera:
Lee tres x de la cinta y coloca a cada una de ellas comenzando desde el tope en el Stack o PILA.
Cuando lea una **y** de la cinta va a una acción POP:
Si lee una y ó blanco de la pila va a un estado de NO ACEPTACION REJECT
sino lee y extrae una x de la PILA . Lee tres y de la cinta y extrae tres x de la pila hasta leer un blanco de la cinta.
Cuando lee el blanco de la cinta va a un nuevo estado POP de lectura de la PILA si acá también lee un blanco, lo extrae pasa a un estado final de ACEPTACION. Si de la PILA leyera un x ó y, va a un estado de detención REJECT.
Si se llegó el estado de aceptación ACCEPT la cadena **xxxyyy** ha sido aceptada

El conjunto de palabras aceptado por este APPD es. { x^n y^n con n >=0}

6.9. MAQUINA DE TURING (MT)

Reconocedora de palabras que derivan de las GSR

Definición de **una MT** por sus elementos

1 Un alfabeto de letras de entrada de la cinta **ɣ** de la cinta, el cual no puede tener símbolos blanco ⍉.

2 Una cinta infinita dividida en celdas. Comienza con la celda i

3 Un cabezal lector escritor de la cinta. Siempre comienza a leer desde la celda i, sise le da la orden de moverse a la izquierda de la celda i falla.

4 Un alfabeto **Ψ** de caracteres que pueden ser impresos en la cinta por el cabezal lector escritor. Se da la siguiente relación de inclusión:

Ψ puede incluir a ɣ

El cabezal lector escritor puede escribir un espacio en blanco, lo denominaremos borrar.

5 Un conjunto finito de estados

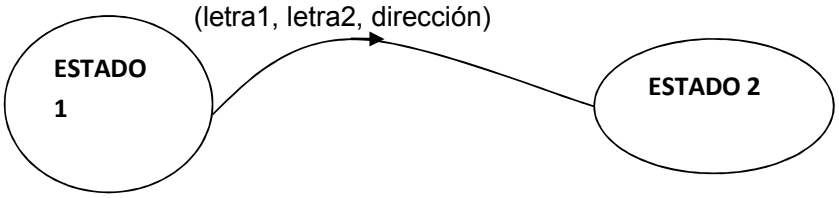
Un estado de comienzo llamado START, al cual se puede volver a ingresar

Estado de detención HALT, puede haber uno o ninguno

Demás estados del conjunto de estados que se reconocen por un número.

6 Un programa que es un conjunto de reglas que indica todos los pasos a seguir como por ejemplo leer, mover, grabar.

El programa se representa por una serie de aristas etiquetadas de la siguiente manera:



Donde estado 1 y estado 2, son dos estados independientes.

En la arista etiquetada **letra1,** carácter que lee el cabezal de acuerdo a la celda que este apuntando, puede ser un espacio en blanco, o pertenecer a **ɣ** o a **Ψ.**

letra2: carácter en blanco o pertenece a **Ψ**. Es el carácter que el cabezal imprime en la cinta antes de abandonarlo.

dirección: el cabezal se mueve a la derecha, R ,o a la izquierda L.

Fallas o terminación de la ejecución:

1-Cuando el cabezal está en la primera celda y se trata de mover el cabezal hacia la izquiderda.

2-Cuando se lee una letra que no ofrece ningún camino hacia otro estado.

Terminación de la ejecución exitosamente

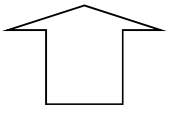
En este caso se debe llegar a un **estado HALT**, se dice entonces que la palabra en la cinta de entrada es aceptada por la MT.

Por definición todas las MT **son deterministas**. Esto significa que de un estado q cualquiera salgan dos o más aristas etiquetadas con la misma primera letra, a estados diferentes.

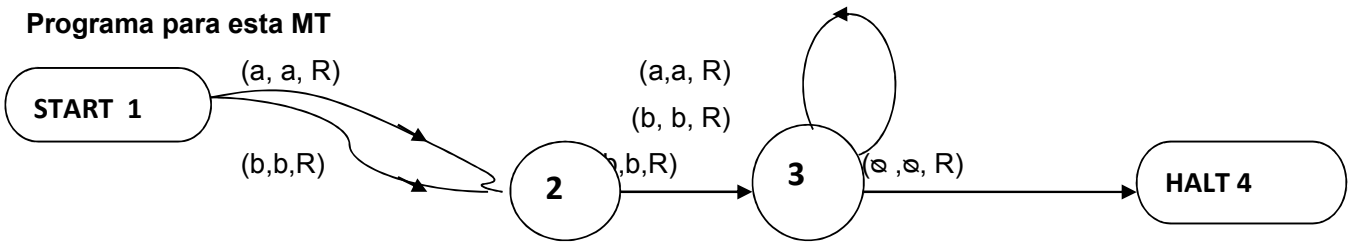
Reconocimiento de un lenguaje regular

Cinta de una MT que correrá sobre la entrada **aba**

a	b	a	⌀	⌀



Programa para esta MT



El lenguaje regular aceptado por esta máquina es : (a| b) b (a|b)*

6.10. AUTOMATA LINEALMENTE ACOTADO (ALA)

Un **autómata linealmente acotado**, abreviadamente **ALA** es similar a una MT. LA diferencia con estas maquinas es que en los ALA las cintas NO son infinitas. Se utilizan para lenguajes generados por GDC Un autómata linealmente acotado es una máquina de Turing cuya cinta está formada solamente por celdas de M.n de largo, donde la longitud n es la secuencia de la entrada y M es una constante asociada al *autómata linealmente-acotado* particular, es decir la cantidad de cinta que el autómata permite usar. La entrada de la cadena en la cinta es el único espacio que la cinta permite usar, todo el proceso se hace entre los marcadores del extremo. Los ALA son más poderosos que los APDND, pero menos potente que las MT

Componentes de un ALA Un autómata linealmente acotado está formado por los siguientes componentes:

ALA: {E, A, B, δ, q0, F, #, $} donde: E Espacio finito de la cinta
- A Alfabeto de entrada
- B Alfabeto de la cinta
- δ Función de transición
- q0 Estado inicial
- F Conjunto de estados finales
- # Símbolo de inicio de cinta
- $ Símbolo de fin de cinta