III

AUTÓMATAS FINITOS.

Palabras clave : Automátas finitos determinísticos, no deterministicos, AFD,AFND, reglas de Thompson, algoritmo de construcción de subgrupos, algoritmo de particiones.AFD óptimo, AFD reducido.

3.1 INTRODUCCIÓN.

Habíamos establecido en los capítulos anteriores, que un analizador léxico reconocía tokens, mediante un monitoreo de izquierda a derecha del programa fuente. Para hacer esta tarea menos difícil, utilizábamos las expresiones regulares para la especificación de los patrones o reglas que cumplen los tokens.
Los *autómatas finitos* son las herramientas empleadas como *reconocedores de tokens,* fig. 3.1.

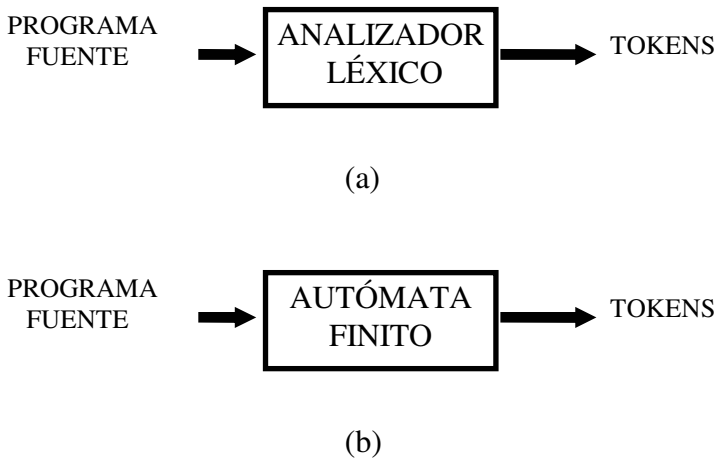


Fig. 3.1 a) Análisis léxico, b) Interior del analizador léxico.

Un autómata finito es capaz de reconocer un *conjunto regular*, es decir, un conjunto de cadenas denotado por cualquier expresión regular. Recordemos que una expresión regular denota a un lenguaje regular.
Un autómata finito es un reconocedor para un lenguaje, su programación no es una tarea compleja, su entrada es una cadena **x** y responde **"si"** si **x** es una sentencia del lenguaje, **"no"** de otra manera, fig. 3.2.

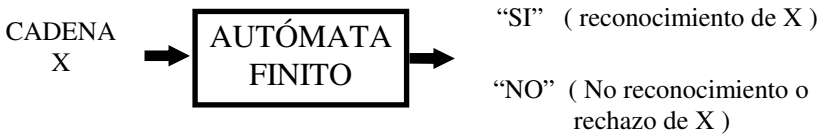


Fig. 3.2. Entrada y salida de un autómata finito.

Los autómatas finitos se clasifican en :
a) Determinísticos.
b) No Determinísticos.

3.2 AUTÓMATA FINITO DETERMINÍSTICO (AFD).

Es un modelo matemático que consiste de :

- Un conjunto de estados, denominado **S**.
- Un conjunto (*alfabeto*) de símbolos de entrada, denominado Σ.
- Una función de transición **move** que mapea un par **P (s , a)** a un estado **t.**
  s y **t** son estados contenidos en **S**, **a** es un símbolo de entrada.
- Un estado de inicio, denotado por s_0.
- Un conjunto de estados de aceptación (finales), denotado por **F**.

Además, un autómata finito determinístico AFD debe cumplir con las siguientes características :
a) No hay transiciones etiquetadas por $\in$.
b) Para cada estado **s** y un símbolo de entrada **a**, existe a lo más un arco etiquetado por **a** saliendo de **s**.

Ejemplo 3.1. *El AFD que reconoce al token id identificador en Pascal es mostrado en la fig. 3.3. La definición regular es :*

Letra $\Rightarrow$ *[A-Za-z]*
Dig $\Rightarrow$ *[0-9]*
Sub $\Rightarrow$ *_*
Id $\Rightarrow$ *Letra (Letra | Dig | Sub) **

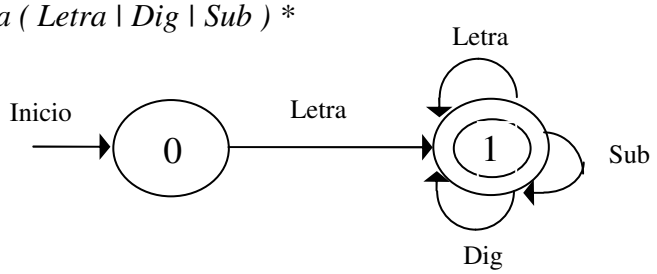


Fig 3.3 AFD para el token *id* en Pascal.

Un autómata es una representación gráfica que muestra el proceso de reconocimiento de una cadena de entrada. La simbología utilizada es simple :

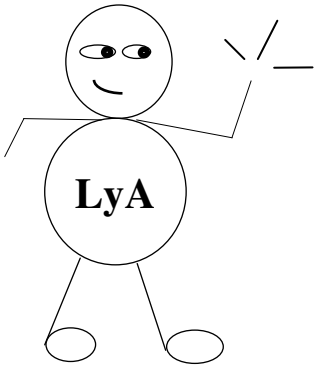
(n) Un círculo representa un estado **n**, donde **n** es un número natural o bien una letra, generalmente.

a → Un arco representa la lectura de un símbolo **a** en la entrada. Transición entre estados.

inicio → (s) Estado de inicio **s**. **s** es generalmente 0 (cero).

(f) Estado de aceptación **f**.

De acuerdo a la notación anterior, dispongámonos a obtener los componentes del *AFD* de la fig. 3.3.



1. El conjunto de estados del autómata AFD es :

$$S = \{ 0 , 1 \}$$

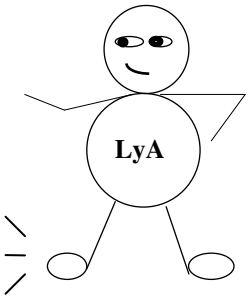
Dos estados... !!!

2. El conjunto o alfabeto de símbolos de entrada, se obtiene de los arcos $\xrightarrow{a}$ del *AFD*.

$$\Sigma = \{ \underbrace{A, B, \ldots , Z, a, b, \ldots , z}_{\text{Letra}}, \underbrace{0, 1, \ldots , 9}_{\text{Dig}} , \underbrace{\_}_{\text{Sub}} \}$$

Es decir el alfabeto Σ es el conjunto de las *letras* mayúsculas y minúsculas, unidas a los *dígitos* y el símbolo de *subrayado* (_).

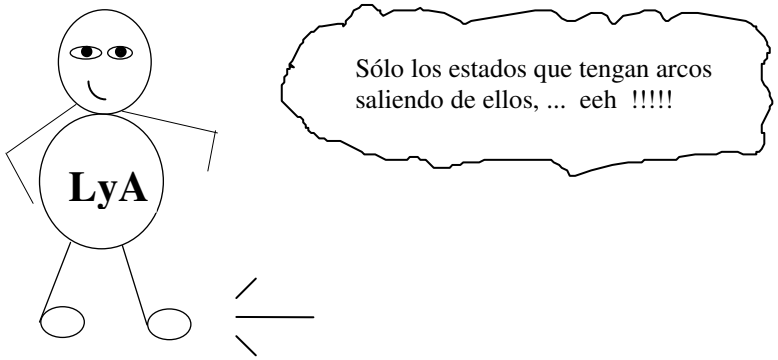
3. La función de transición *move* es realmente una tabla de transición, donde los renglones son los *estados* y las columnas son *símbolos de entrada*.



		Símbolos en la entrada		
		Letra	Dig	Sub
Estados	0	1	-	-
	1	1	1	1

Fig. 3.4 Función *move* para el AFD que reconoce al token *id* en Pascal.

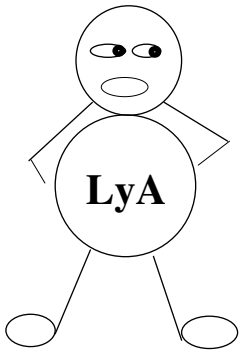
La construcción de la tabla de transición -función *move*- se inicia identificando los estados que tienen al menos un arco que "salga" de ellos. Para nuestro ejemplo el estado 0 (cero) tiene un arco etiquetado por *Letra* que "sale" de él, y el estado 1 tiene 3 arcos: *Letra*, *Dig* y *Sub*. Los estados que tengan esta característica son añadidos como los renglones en la tabla.



En las columnas debemos ubicar a todos los posibles símbolos del alfabeto que pueden ocurrir, al efectuar una lectura en la entrada (cadena). En nuestro caso son: *Letra*, *Dígito* y *Subrayado*.

En un AFD con columnas c_1 , c_2 , ... , c_k donde $\underline{k}$ es el número de posibles entradas, se debe cumplir:

$$c_i \cap c_j = \emptyset, \quad \text{para } i \neq j$$



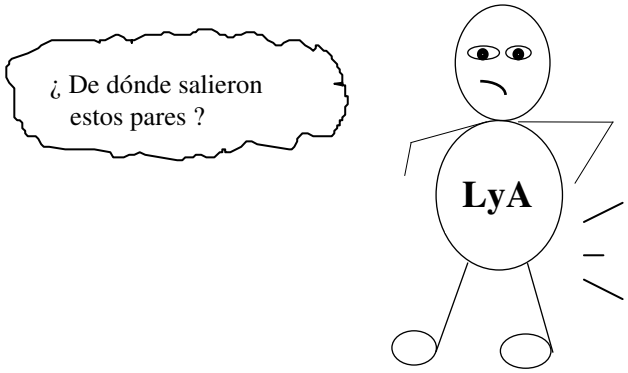
Lo anterior es sumamente importante, dado que si no se cumple $\mathbf{c_i \cap c_j = \emptyset}$ podríamos tener un autómata finito *no determinístico.*

La tabla de transición de la fig. 3.4 si cumple con la regla :

Letra $\cap$ Dig = $\emptyset$
Letra $\cap$ Sub = $\emptyset$
Dig $\cap$ Sub = $\emptyset$ **Ø** es el conjunto vacío.

Los pares *p (s , a)* representados en la tabla se listan enseguida:

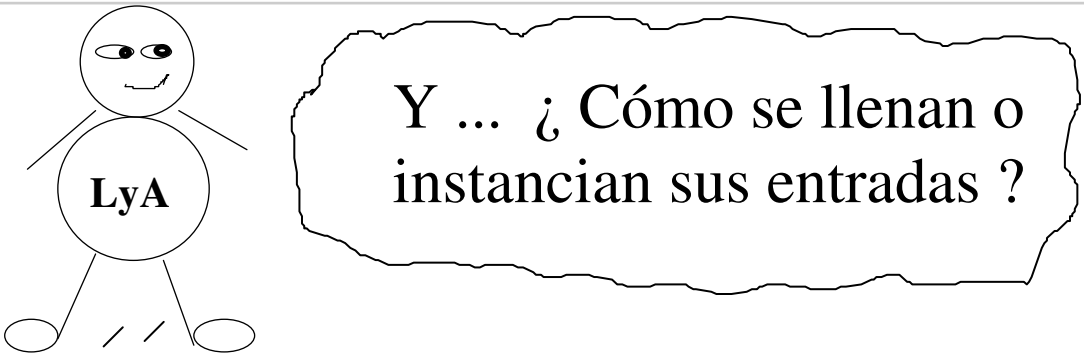
(0 , Letra) → 1
(0 , Dig) → Error
(0 , Sub) → Error
(1 , Letra) → 1
(1 , Dig) → 1
(1 , Sub) → 1



La tabla antes del mapeo de los pares *p (s , a)* a un estado **t**, estaba vacía, es decir :

Simbolos en la entrada

		Letra	**Dig**	**Sub**
Estados	0			
	1			



Pues precisamente del diagrama del AFD de la fig. 3.3. Por ejemplo, el *par (0 , Letra)* se puede leer : "Si el estado es el 0, y la lectura en la entrada es una letra, el autómata se **moverá** al estado 1". O sea que :

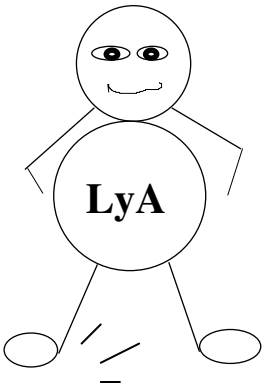
move (0 , Letra) = 1

Esta transición es representada por el arco etiquetado por *Letra*, y que "sale" del estado 0 hacia el estado 1, en el diagrama del AFD. El resultado de la función *move (s , a)* para un automata finito detreminístico es **un solo estado** ó bien **ninguno** (caso de error).

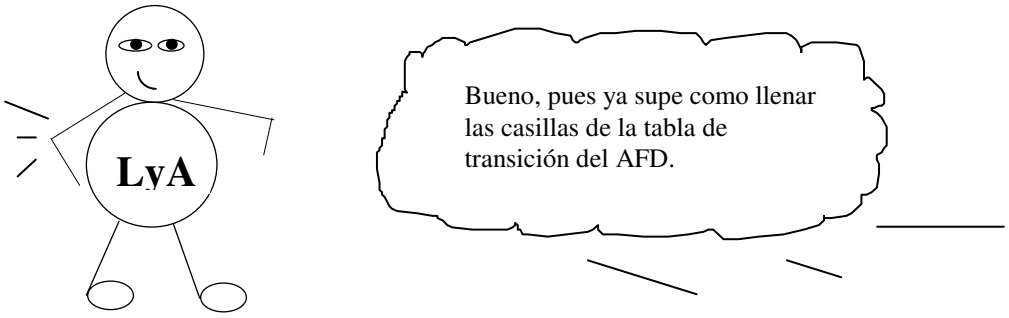
move (s,a) = un solo estado.

NINGÚN ESTADO equivale a un error.

..... para un *AFD*.



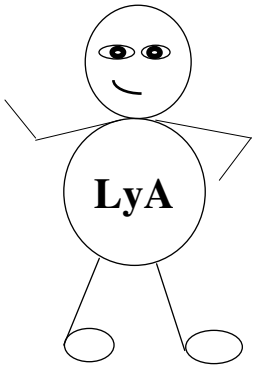
Los move's que producen error, son los pares: *(0 , Dig) y (0 , Sub).*
move (0 , Dig) = error, *move (0 , Sub)* = error, ya que en el AFD no hay arco que salga del estado 0 (cero) y que sea etiquetado ya sea por *Dig* o por *Sub*.



El estado de inicio es el estado 0 y siempre se indica con un arco llegando a dicho estado.

4. $s_0 = 0$

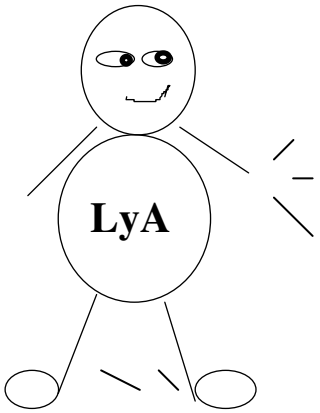
Estado de inicio del AFD.



Sólo tenemos un estado de aceptación y es el denotado con doble círculo : ***1***. Así, el quinto componente *F* es:

$$F = \{ 1 \}$$

La notación de conjuntos es debido precisamente a que *F* es un conjunto.



Las características que además debe cumplir el *AFD* son :

a) No arcos $\in$. Si se cumple, dado que no existe un solo arco etiquetado $\in$ con en el diagrama del AFD.
b) De los estados 0 y 1, se observan *sólo arcos etiquetados en forma única.* Nunca se repite una etiqueta para dos o más arcos saliendo de un estado.

Existe un algoritmo para simular el funcionamiento de un *AFD*, al efectuar el reconocimiento o rechazo de una cadena. Fig. 3.5.

Utilicemos el algoritmo de la fig. 3.5 para reconocer la cadena *iCont1* con el AFD del ejemplo 3.1. La ejecución paso a paso se muestra en la tabla de la fig. 3.6, donde las columnas son variables y sus valores se indican, de acuerdo a la instrucción que se ejecuta en secuencia.

Entrada

Cadena x terminada en el caracter *eof*.

D (autómata AFD) con estado de inicio s_0 y *F* conjunto de estados finales.

Proceso

```
s = s0
c = NextChar ( )
while ( c ≠ eof ) do
   s = move ( s , c )
   c = NextChar ( )
 endwhile
 if ( s está en F )
    then
      return 'si'
    else
      return 'no'
```

Salida

Respuesta 'si' si *D* acepta a *x*, de otra manera la respuesta es 'no'.

Fig. 3.5. Carta EPS, del algoritmo que simula a un AFD.

s_0	F	s	c	move (s , c)	
0	{ 1 }	0	i		s = s_0 c = NextChar()
		1		move (0 , i)	while (c ≠ eof) do
			C		s = move (s , c)
		1		move (1 , C)	c = NextChar()
			o		endwhile
		1		move (1 , o)	
			n		
		1		move (1 , n)	
			t		
		1		move (1 , t)	
			1		
		1		move (1 , 1)	

Fig. 3.6. Reconocimiento de la cadena *iCont1*.

La función *NextChar( )* retorna el siguiente caracter de la cadena de entrada *x*.

Dado que $s = 1$, entonces si pertenece a F, por lo tanto el algoritmo retorna un *"si"*.

De la tabla fig. 3.6 podemos ver que :

move (0 , i) = 1
move (1 , C) = 1
move (1 , o) = 1
move (1 , n) = 1
move (1 , t) = 1
move (1 , 1) = 1

¿ Cómo son obtenidos estos *move's* ?

De la tabla de transición del AFD Fig. 3.4. Por ejemplo el *move (0 , i)* = 1, se lee : "si el autómata se encuentra en el estado *0* y llega una letra -en este caso la *i*-, el autómata pasa al estado *1*. El *move (1 , 1)* = 1, se busca en el renglón cuyo estado es *1* y la columna *Dig* (dígito).

Ejemplo 3.2. Existe otro AFD para reconocer el token *id* del ejemplo 3.1 que no es óptimo, ya que tiene más estados y transiciones que el anterior.

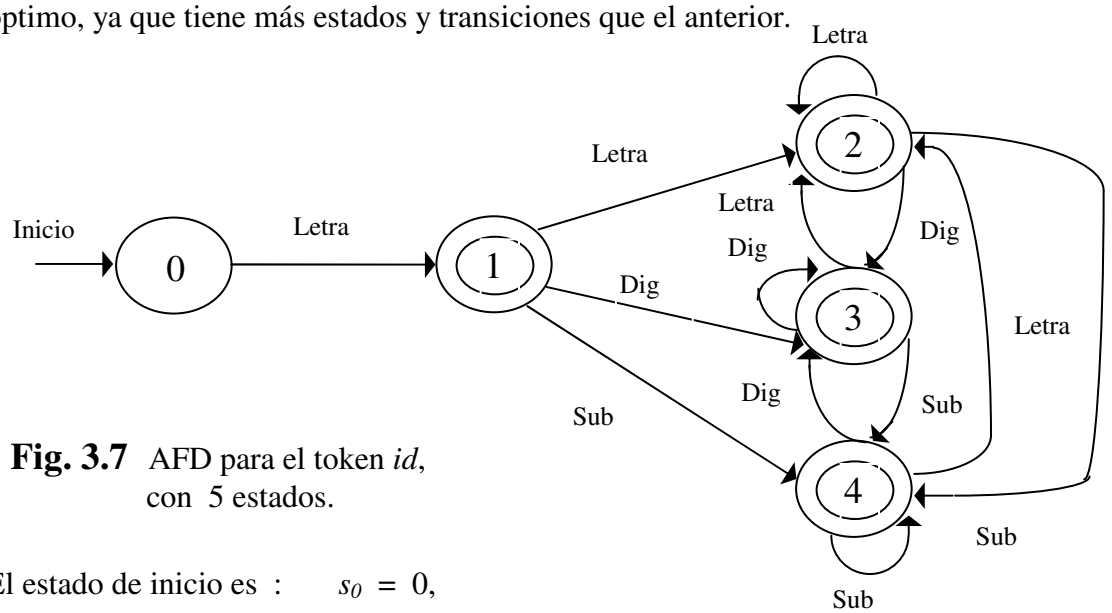


Fig. 3.7 AFD para el token *id*, con 5 estados.

El estado de inicio es : $s_0 = 0$,

Los estados de aceptación son 4 : $F = \{ 1, 2, 3, 4 \}$

El alfabeto es el mismo, es decir : $\Sigma = \{ A, B, \ldots , Z, a, b, \ldots , z, 0, 1, \ldots , 9 , \_ \}$

El conjunto de estados *S* del AFD, tiene 5 elementos : $S = \{ 0, 1, 2, 3, 4 \}$

La tabla de transición -función *move*- tiene 5 renglones (todos los estados, tienen al menos un arco saliendo de ellos) y 3 columnas :

		Símbolos en la entrada		
		Letra	**Dig**	**Sub**
Estados	0	1	-	-
	1	2	3	4
	2	2	3	4
	3	2	3	4
	4	2	3	4

move (0 , Letra) = 1	*move* (2 , Sub) = 4
move (0 , Dig) = Error	*move* (3 , Letra) = 2
move (0 , Sub) = Error	*move* (3 , Dig) = 3
move (1 , Letra) = 2	*move* (3 , Sub) = 4
move (1 , Dig) = 3	*move* (4 , Letra) = 2
move (1 , Sub) = 4	*move* (4 , Dig) = 3
move (2 , Letra) = 2	*move* (4 , Sub) = 4
move (2 , Dig) = 3	

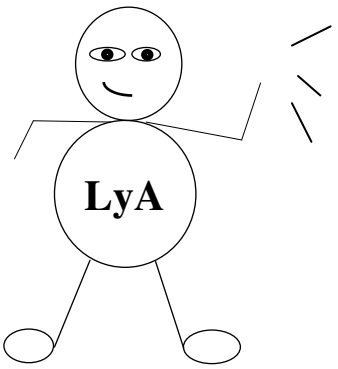
En la fig. 3.8 se muestra el trazo del algoritmo que simula al AFD, en el reconocimiento de la cadena **X11A_2**.

s_0	F	c	s	move (s , c)	

0	{ 1, 2, 3, 4 }	x	0		s = s_0
					c = NextChar()
					while (c ≠ eof) do
					s = move (s , c)
					c = NextChar()
					endwhile

			1	move (0 , x)	
		1			
			3	move (1 , 1)	
		1			
			3	move (3 , 1)	
		A			
			2	move (3 , A)	
		_			
			4	move (2 , _)	
		2			
			3	move (4 , 2)	

Fig. 3.8. Simulación del AFD con *F = { 1, 2, 3, 4 }*



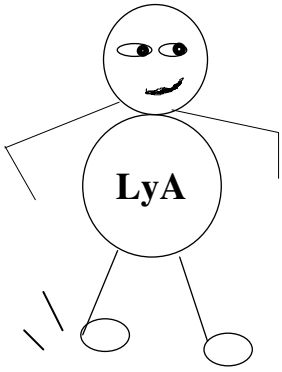
Ya que *s* = 3 y *s* pertenece a *F*, el AFD retorna un ..... *"si"*

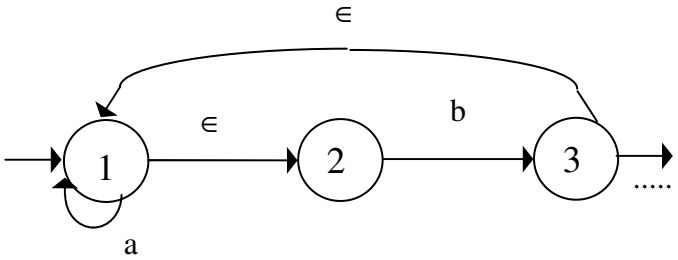
3.3 AUTÓMATA FINITO NO DETERMINÍSTICO (AFND).

Un autómata finito no determinístico es un modelo matemático que consiste de :

1. Un conjunto de estados, **S**.
2. Un conjunto de símbolos de entrada, ∑ (alfabeto).
3. Una función de transición denominada **move**, que mapea pares, *p (s , a)* hacia un conjunto de estados. **s** es un estado y **a** es un símbolo en la entrada.
4. Un estado de inicio denotado por s_0.
5. Un conjunto de estados de aceptación, denotado por **F**.

Además, un *AFND* tiene como característica que lo diferencia de un *AFD*, permitir el uso de arcos etiquetados por el símbolo $\in$.

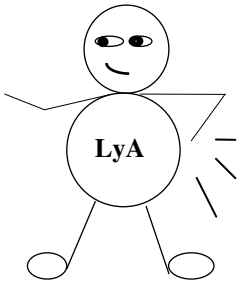




Arcos $\in$

La característica de donde toma el nombre un *AFND*, consiste en que su función de transición **move**, mapea los pares *p (s , a)* hacia un *conjunto de estados*.

Revisa el punto 3 para un AFD y vas a encontrar que el autómata determinístico, mapea los pares p(s,a) .... a un sólo estado. !!!!



¿ Qué significa lo anterior ?. Significa que un AFND que se encuentra en un estado **s** y para un símbolo en la entrada **a**, pueden existir más de un arcos etiquetados con el símbolo **a** saliendo del estado **s**, Fig.3.9. *Por lo tanto no está determinada la transición*.

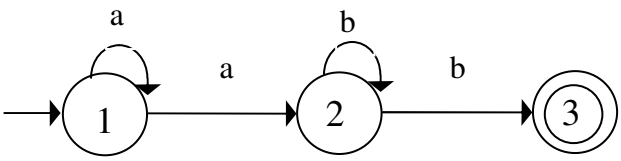


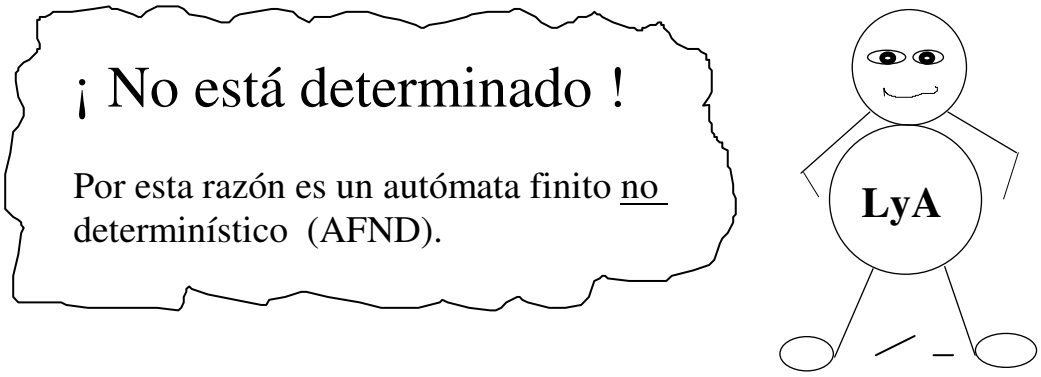
Fig. 3.9 AFND para $a^+ b^+$

En la Fig. 3.9 se muestra el AFND para la expresión regular a^+b^+. La función *move (1 ,a)* se define como :

move (1 , a) = { 1 , 2 } y *move* (2 , b) = { 2 , 3 }

¿Hay dos estados a donde el autómata puede efectuar una transición ?

En ambos casos la función *move* nos mapea a 2 estados ¿ A qué estado el autómata efectúa la transición ?



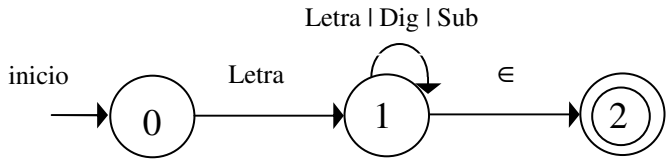
Los componentes del AFND de la Fig. 3.9 son :
El conjunto de estados $S = \{ 1, 2, 3 \}$, estado inicial $s_0 = 1$ y sólo hay un estado de aceptación $F = \{ 3 \}$. El alfabeto de entrada es $\Sigma = \{ a, b \}$. S, s_0, F y Σ son obtenidos directamente del diagrama que representa el AFND.
La tabla de transición es :

		Símbolos en la entrada	
		a	**b**
Estados	1	{ 1 , 2 }	-
	2	-	{ 2 , 3 }

move (1 , a) = { 1 , 2 }
move (1 , b) = Error
move (2 , a) = Error
move (2, b) = { 2 , 3}

Podemos apreciar que el estado 3 no se incluyó en los renglones de estados, ya que no existen arcos que "salgan" de él, o sea, *move (3 , a)* = Error y *move (3 , b)* = Error.

Ejemplo 3.3 *Veamos el correspondiente AFND para el token id en Pascal, de los ejemplos 3.1 y 3.2.*



Los componentes del AFND son :

S = { 1, 2, 3 }
s_0 = 0
Σ = { A, B, ... , Z, a, b, ... , z, 0, 1, ... , 9 , _, ∈ }
F = { 2 }

La función de transición *move* :

Símbolos en la entrada

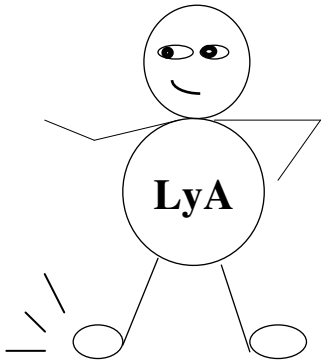
Estados		**Letra**	**Dig**	**Sub**	∈
	0	{ 1 }	-	-	-
	1	{ 1 }	{ 1 }	{ 1 }	{ 2 }

Se ha incluído el símbolo ∈ en la tabla de transición, debido a la característica del AFND que permite arcos -transiciones- etiquetadas por ∈.
No podemos utilizar el algoritmo de la fig. 3.5 para simular el AFND, pero si podemos efectuar el trazo de la función *move* cuando se reconoce una cadena. Por ejemplo, si la cadena es *iCont*, tenemos lo siguiente :

$$0 \xrightarrow{i} 1 \xrightarrow{C} 1 \xrightarrow{o} 1 \xrightarrow{n} 1 \xrightarrow{t} 1 \xrightarrow{\in} 2$$

"si" acepta la cadena *iCont*

Observa que para este ejemplo, el mapeo de los pares *p(s,a)* es siempre hacia un sólo estado. En este caso **si** está determinada la transición, pero el autómata es no determinístico, debido al arco ∈ del estado 1 al estado 2.



3.4 AUTÓMATAS FINITOS Y EXPRESIONES REGULARES.

Existen algoritmos que relacionan la especificación de tokens -expresiones regulares-, con el reconocimiento de éstos -autómatas finitos-. Es posible dada una expresión regular obtener el AFD que reconozca las cadenas del lenguaje denotado por la expresión regular. También es posible obtener el AFND que reconozca el lenguaje representado por dicha expresión regular.

El algoritmo que permite construir el autómata finito determinístico está fuera del alcance de estas notas (el alumno no tiene los prerequisitos para su estudio en este curso).
Sin embargo, el algoritmo utilizado para la construcción del autómata finito no determinístico AFND, es relativamente sencillo de aplicar, ya que se basa en reglas simples. Existen muchas variantes de este algoritmo denominado *"Algoritmo de Thompson"*.

Este algoritmo es *dirigido por sintáxis*, es decir, usa la estructura sintáctica de la expresión regular para guiar el proceso de construcción del autómata AFND. En la Fig 3.10 se muestra la carta Entrada-Proceso-Salida (EPS) para el algoritmo de construcción de Thompson.

Entrada
Expresión regular **r** definida sobre un alfabeto Σ.

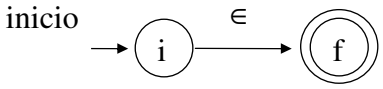
Proceso
Primero, reconocemos las subexpresiones que constituyen a **r**.
Usando las reglas (1) y (2), construímos los *AFND's* para cada símbolo básico en **r**.
Guiados por la estructura sintáctica de la expresión regular **r**, combinamos estos *AFND´s* de manera inductiva usando la regla (3) hasta obtener el *AFND* para la expresión regular **r**.

Salida
Un *AFND* que reconoce al lenguaje *L (r).*

Fig. 3.10 Carta EPS para el algoritmo de construcción de Thompson.

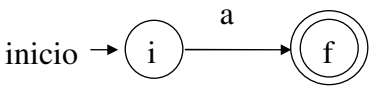
Las reglas a las que hace mención el algoritmo de Thompson son las siguientes :

1. Para el símbolo $\in$, construir el AFND :



i es el nuevo estado inicial, y **f** es el nuevo estado de aceptación. Este AFND reconoce a $\{ \in \}$.

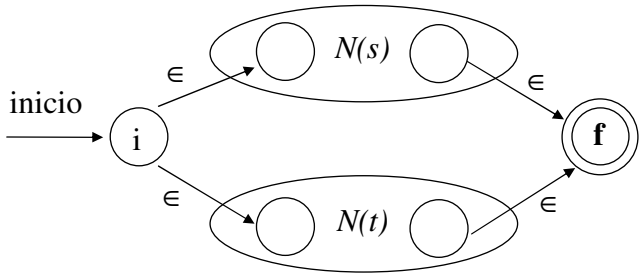
2. Para cualesquier símbolo **a** del alfabeto ∑, construir el AFND :



De nuevo, **i** es el nuevo estado inicial, y **f** es el nuevo estado de aceptación. Este autómata reconoce { a }.

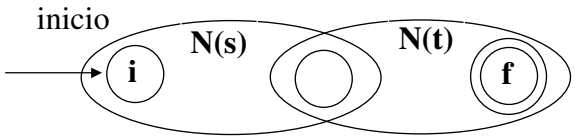
3. Supongamos que **N(s)** y **N(t)** son AFND's para las expresiones regulares **s** y **t**, respectivamente.

a) Para la expresión regular ***s* | *t*** (alternancia), construir el siguiente AFND, **N(s|t)** :



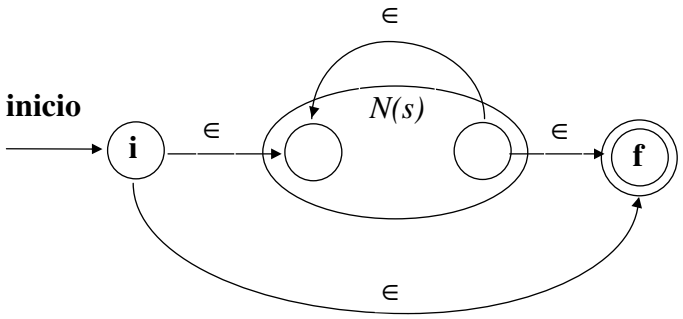
i es el nuevo estado inicial, y **f** es el nuevo estado de aceptación. Se añade una transición ∈ desde **i** hacia los estados de inicio de *N(s)* y de *N(t).* Además, se añade una transición ∈ desde los estados de aceptación *N(s)* y de *N(t)* hacia el nuevo estado de aceptación **f**. Los estados de inicio y de aceptación de *N(s)* y de *N(t)* no son los estados de inicio y de aceptación del autómata *N(s|t).* Este AFND reconoce, *L(s)* ***U*** *L(t).*

b) Para la expresión regular **st** (concatenación), construir el AFND, **N(st)** :



El estado de inicio de *N(s)* es ahora el estado de inicio para el AFND *N(st),* y el estado de aceptación de *N(t)* se vuelve el estado de aceptación del AFND, *N(st).* El estado de aceptación de *N(s)* es mezclado con el estado inicial de *N(t);* esto significa que todas las transiciones, desde el estado inicio de *N(t)* son ahora arcos o transiciones desde el estado de aceptación de *N(s).* El nuevo estado que resulta de esta mezcla, pierde su estatus de estado de inicio o aceptación para el nuevo AFND. El AFND así construido, reconoce el lenguaje *L(s) L(t).*

c) Para la expresión regular **s***, construir el AFND, **N(s*)** :



i es un nuevo estado inicial, y **f** es un nuevo estado de aceptación. Con el nuevo AFND se reconoce el lenguaje **(L(s)) ***.

Ejemplo 3.4. *Dada la expresión regular* ***Token*** ***(c|d*)a*** *construir su correspondiente AFND.*

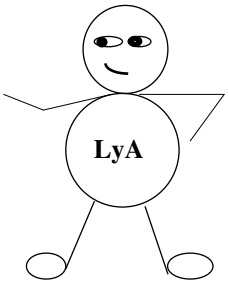
La descomposición sintáctica de la expresión regular consiste de 6 etapas básicamente:

c
d
d *
c | d *
a
(c | d *) a

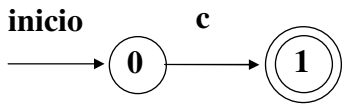
Orden de aplicación de las reglas.

Construiremos 4 AFND's para cada una de las etapas, hasta llegar al AFND que reconoce :

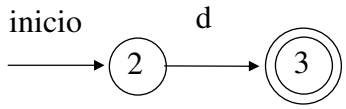
Token → (c | d *) a



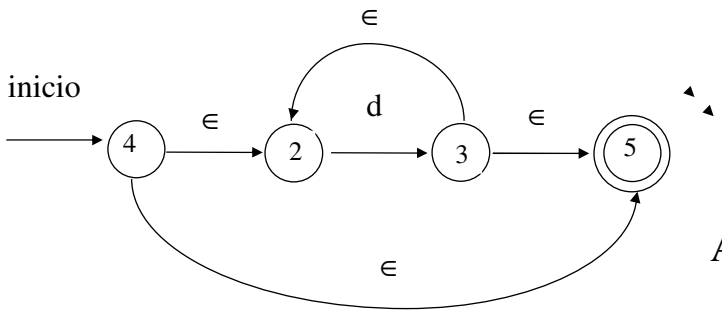
AFND para el símbolo **c**, regla 2 :



AFND para **d***. Primero obtenemos el AFND para el símbolo **d** (regla 2) :

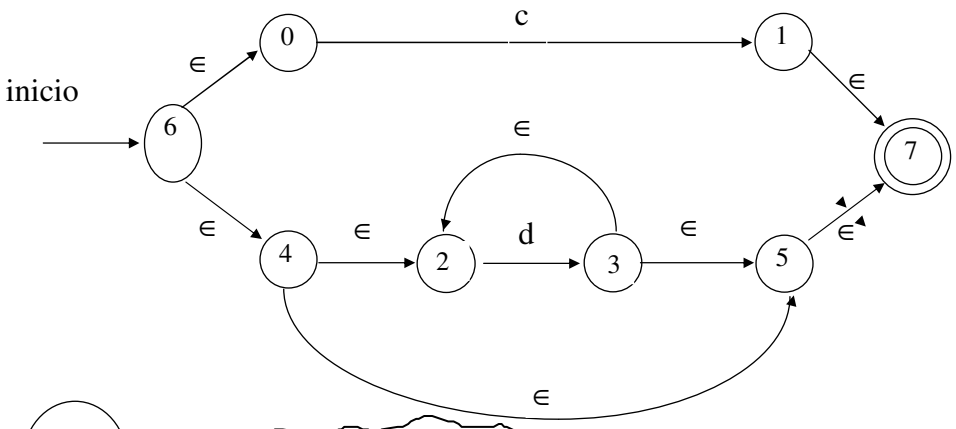


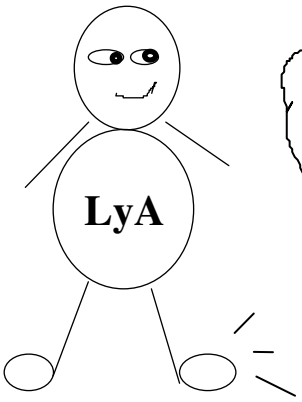
Luego, seguimos con la construcción del AFND para **d*** (regla 3c) :



AFND para d *

Ahora podemos encontrar la tercera etapa, es decir, **c | d ***, la alternancia del símbolo **c** con la cerradura de **d** :

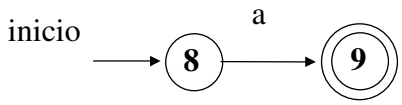




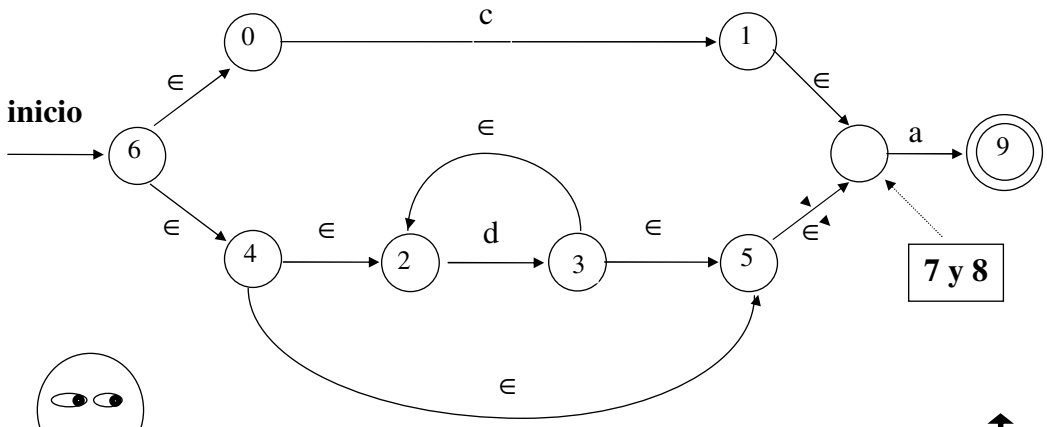
Regla 3 a).

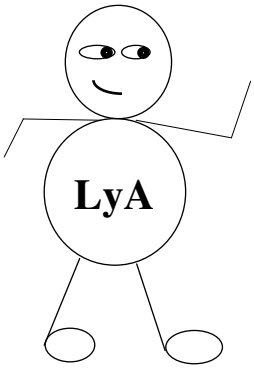
c | d *

Por último, aplicaremos la regla 3b, para concatenar el AFND para **c|d*** con el AFND del símbolo **a**:



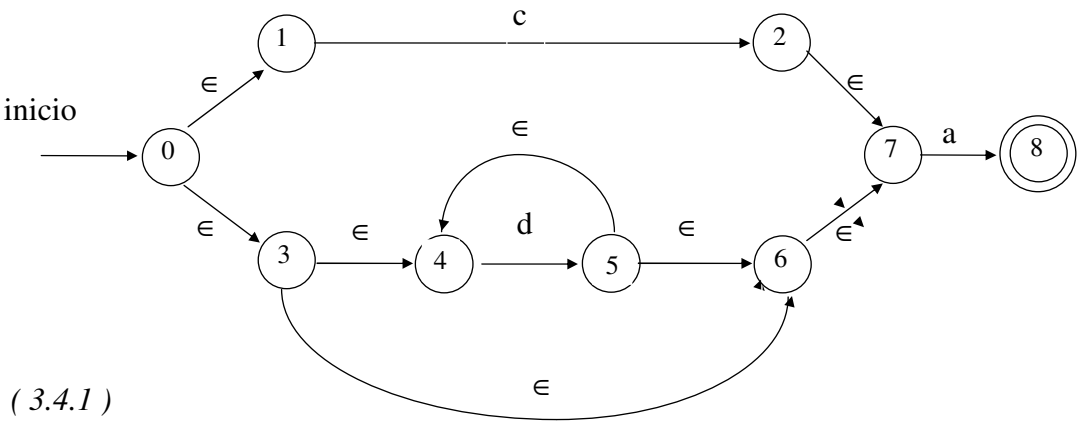
AFND para *a*





Los estados 7 y 8 se unieron para formar uno sólo.
Tienes que ponerle un número de estado único. !!!

Es buena costumbre, reenumerar los estados en orden progresivo ascendente. De acuerdo a ésto, el AFND que reconoce a la expresión regular **Token ⟶ (c | d*) a** es :



(3.4.1)

Cualquier AFND construido con el Algoritmo de Thompson, tiene como característica que existe ***sólo un estado*** de ***inicio*** y ***sólo un estado*** de ***aceptación***. Además, del estado de aceptación **nunca** *"sale"* o existe una transición.

Los componentes para el AFND que reconoce **(c | d*) a** son :

s_0 = 0 ; estado inicial.
F = { 8 } ; estado de aceptación.
S = { 0, 1, 2, 3, 4, 5, 6, 7, 8 } ; conjunto de estados que forman el AFND.
Σ = { $\in$, c, d, a }

Función de transición ***move(s , a)*** :

Símbolos en la entrada

Estados		**a**	**c**	**d**	**$\in$**
	0	-	-	-	{ 1,3 }
	1	-	{ 2 }	-	-
	2	-	-	-	{ 7 }
	3	-	-	-	{ 4,6 }
	4	-	-	{ 5 }	-
	5	-	-	-	{ 4,6 }
	6	-	-	-	{ 7 }
	7	{8}	-	-	-

Ejemplo 3.5. *Dada la expresión regular :*

Letra $\Rightarrow$ *A|B|...|Z|a|b|...|z*
Dig $\Rightarrow$ *0|1|...|9*
Sub $\Rightarrow$ _
Id $\Rightarrow$ *(Letra) (Letra | Dig | Sub) **

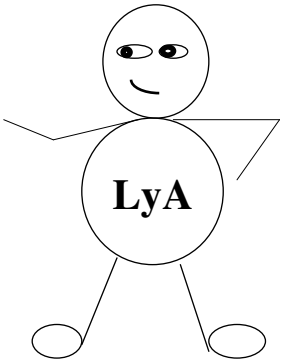
Construir el AFND correspondiente.

Iniciemos observando, que la expresión regular *Id* consiste de la concatenación de la expresión regular *Letra*, con la cerradura de una alternancia de las 3 expresiones regulares : *Letra, Dig y Sub.*

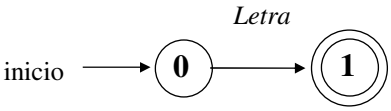
La descomposición sintáctica es la siguiente :

Letra
Dig
Sub
Letra | Dig
Letra | Dig | Sub
(Letra | Dig | Sub) *
(Letra) (Letra | Dig | Sub) *

Tenemos que construir *7 autómatas* para encontrar el AFND deseado.

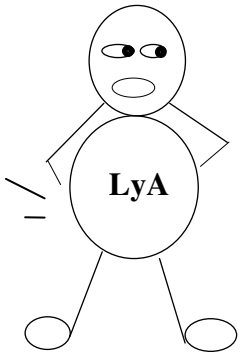


El AFND para letra es construido con la regla 2 :



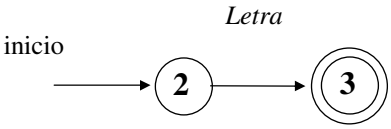
(3.5.1)

El AFND para letra es construido dos veces, en forma separada. En lo anterior hace hincapié el algoritmo de Thompson. Se construye dos veces, dado que la expresión regular *Letra* aparece en dos operaciones :



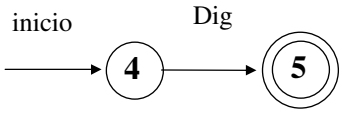
Concatenación	y la alternancia
↓	↓
(Letra)	(Letra | Dig | Sub) *

Así pues, obtengamos el otro AFND para la expresión regular *Letra*.

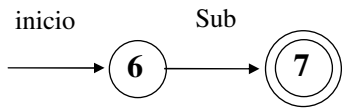


(3.5.2)

La misma regla 2 es aplicada para obtener el AFND para las expresiones regulares *Dig* y *Sub* :

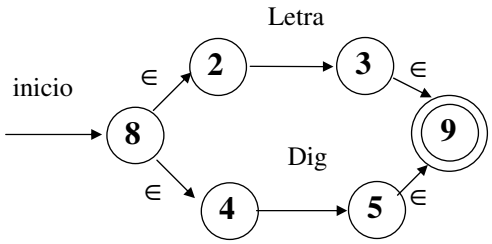


(3.5.3)



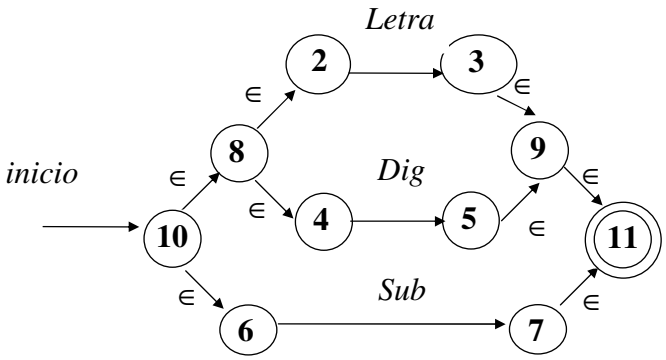
(3.5.4)

La regla 3a) se aplica sobre los AFND's 3.5.2 y 3.5.3 para obtener *Letra* | *Dig* :



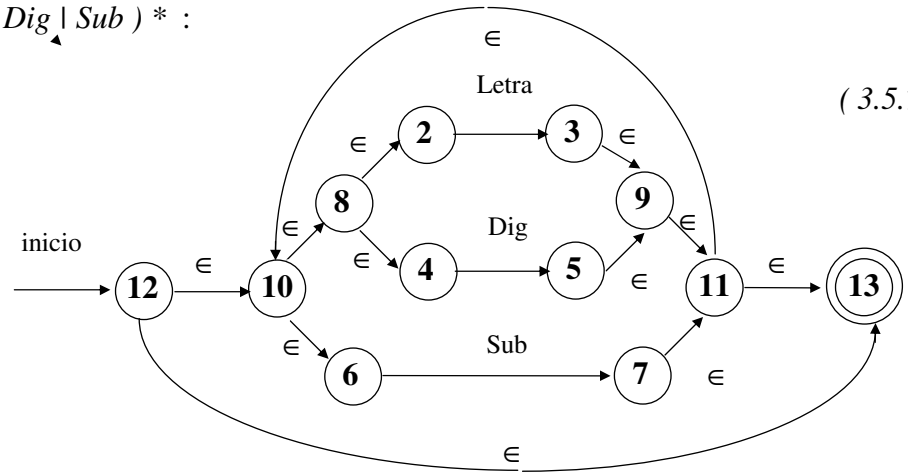
(3.5.5)

El AFND para *Letra* | *Dig* | *Sub* es encontrado con la regla 3 a) y los autómatas (3.5.4) y (3.5.5) :



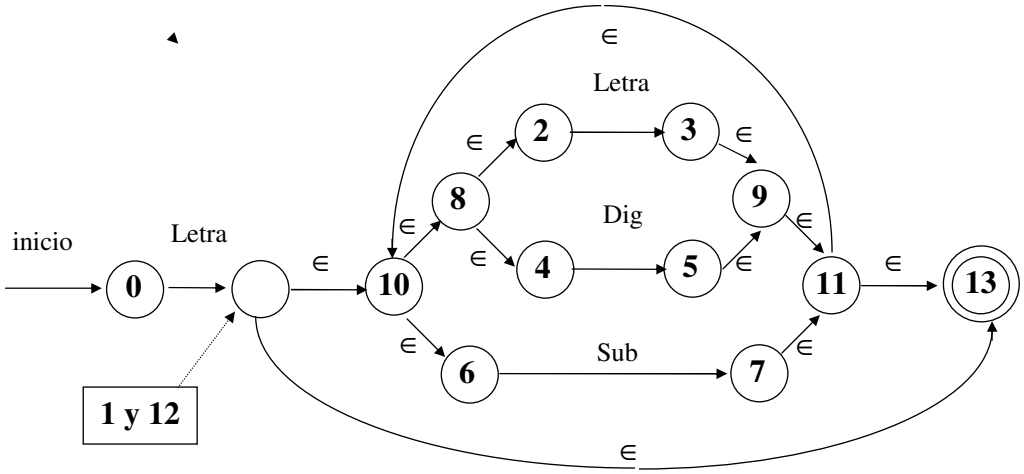
(3.5.6)

Apliquemos ahora la regla 3 c) al AFND 3.5.6, y encontraremos la cerradura *(Letra* | *Dig* | *Sub)* * :

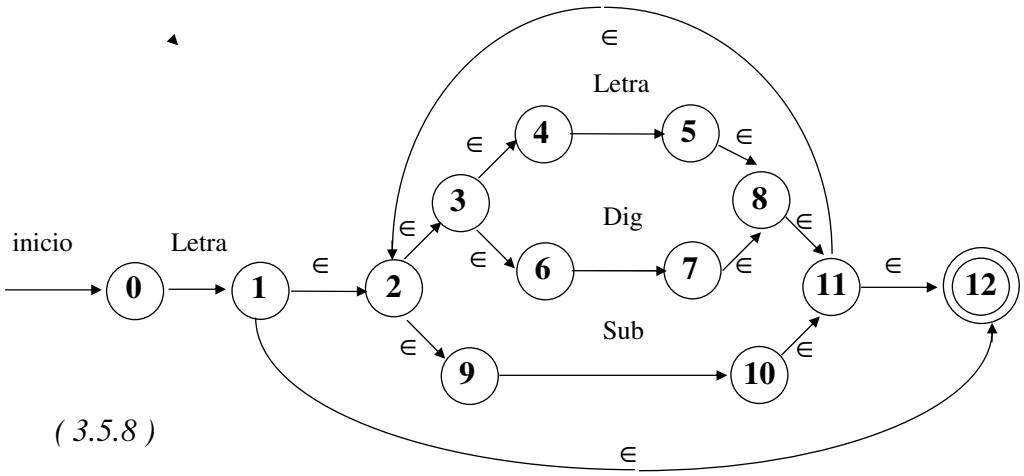


(3.5.7)

Sólo hace falta encontrar la concatenación de *Letra* con *(Letra|Dig|Sub)**. Los AFND's involucrados son el 3.5.1 y 3.5.7. Los estados 1 y 12 se mezclan regla 3 b).



Es recomendable reenumerar los estados del AFND.



(3.5.8)

Los componentes del AFND construido mediante el algoritmo de Thompson :

s_0 = 0
S = { 0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }
Σ = { A, B, ... , Z, a, b, ... , z, 0, 1, ... , 9 , _ }
F = { 12 }

Función *move (s,a) :*

Símbolos en la entrada

Estados		**Letra**	**Dig**	**Sub**	**∈**
	0	{ 1 }	-	-	-
	1	-	-	-	{ 2,12 }
	2	-	-	-	{ 3,9 }
	3	-	-	-	{ 4,6 }
	4	{ 5 }	-	-	-
	5	-	-	-	{ 8 }
	6	-	{ 7 }	-	-
	7	-	-	-	{ 8 }
	8	-	-	-	{ 11 }
	9	-	-	{ 10 }	-
	10	-	-	-	{ 11 }
	11	-	-	-	{ 2,12 }

Ejemplo 3.6. *Construir el AFND que reconoce la definición regular :*

Dig → *0 | 1 | ... | 9*

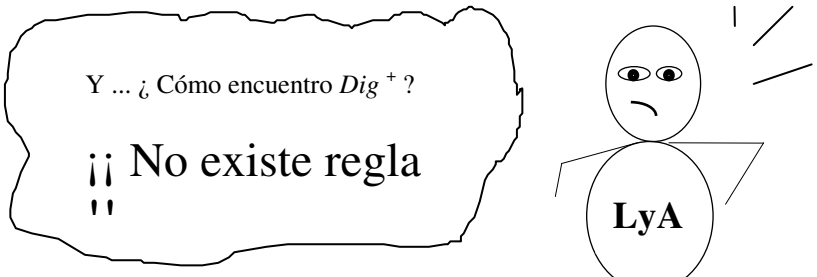
NumEsp → $Dig^+.Dig^+$

NumEsp es una expresión regular que denota al lenguaje formado por las cadenas que son números con punto decimal y al menos una cifra entera y una cifra en la fracción.
La descomposición sintáctica de la expresión regular *NumEsp*, es una concatenación de 3 subexpresiones regulares :

Dig^+ // Cerradura positiva de *Dig*
• // Caracter *punto*
Dig^+ // Cerradura positiva de *Dig*

Tenemos que construir 7 autómatas para llegar a la solución :

Dig
Dig^+
.
Dig^+ .
Dig
Dig^+
Dig^+ . Dig^+



Pero sabemos que :

$\mathbf{r^+ = r\,r^*}$

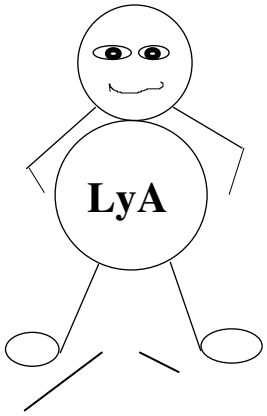
Por lo tanto, podemos descomponer Dig^+ en :

$Dig^+ = Dig\ Dig^*$

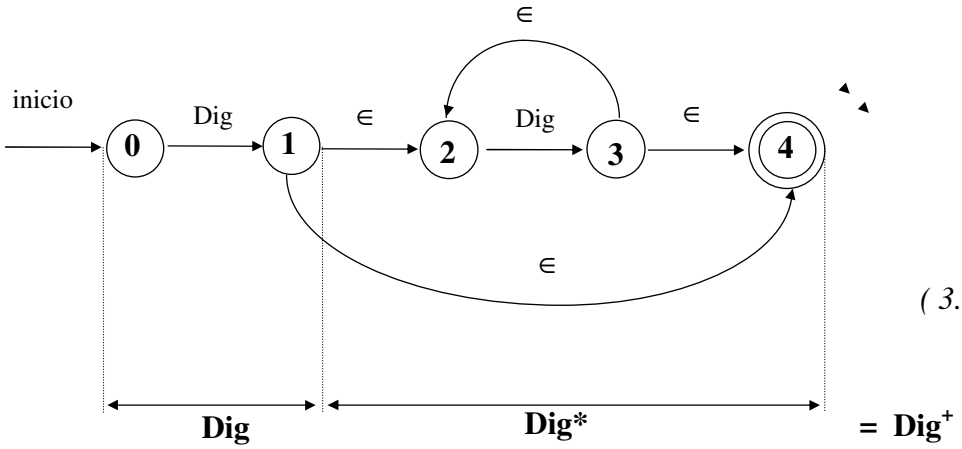
Así, los AFND's que debemos obtener son los siguientes :

Dig
Dig *
Dig Dig * // Dig^+
.
Dig Dig * .
Dig
Dig *
Dig Dig * // Dig^+
Dig Dig * . Dig Dig *

Orden de construcción de los AFND's.



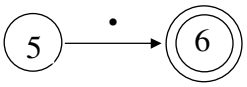
Aplicando reglas 2, 3 c) y 3 b), tenemos el AFND para Dig^+ :

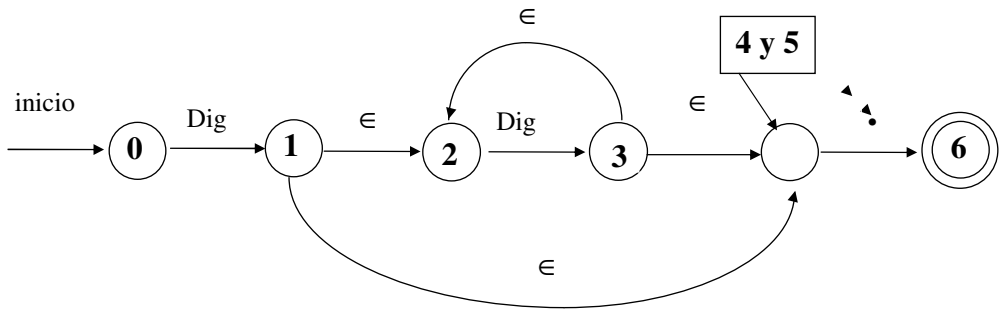


(3.6.1)

La regla 3 b es para $DigDig^*$. Los estados 4 y 5 se mezclan al efectuar la concatenación :

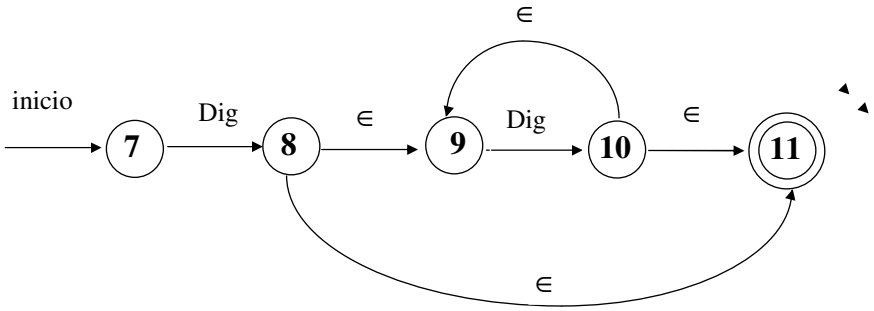
AFND para el símbolo "**.**"





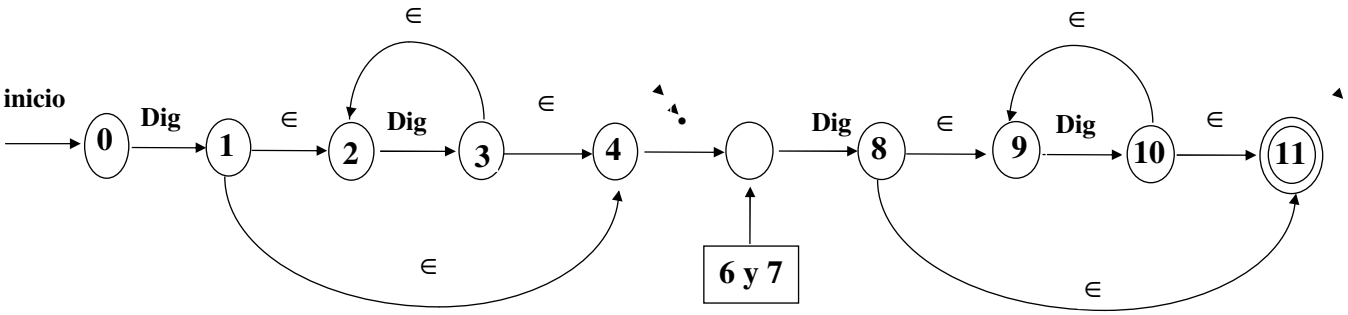
AFND para Dig^{+} **.** *(3.6.2)*

El AFND para *Dig Dig*,* expresión regular que forma el tercer término en *NumEsp* ⟶ $Dig^{+}.Dig^{+}$, es igual que el AFND (3.6.1) salvo la numeración de estados :

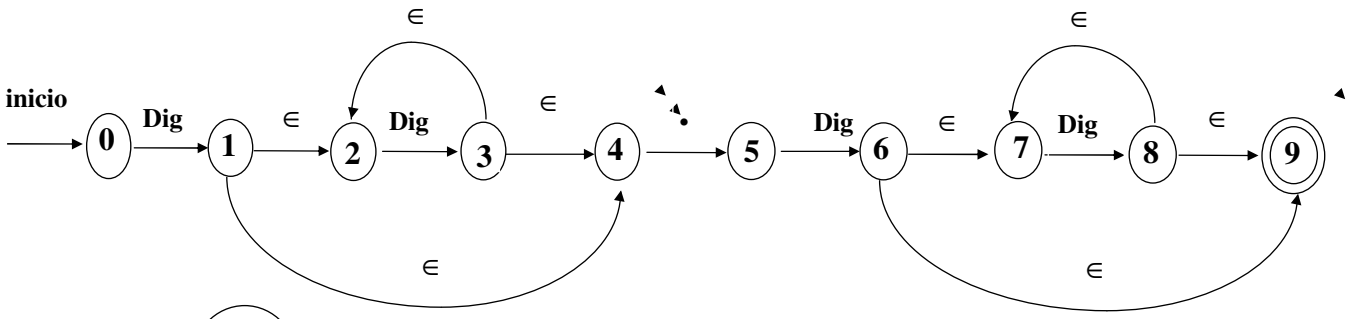


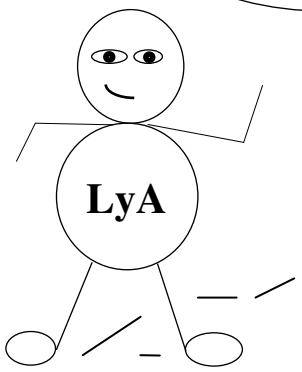
(3.6.3)

Los estados 6 y 7 se mezclan en uno, al aplicar la regla 3 b), para los AFND 3.6.2 y 3.6.3.



El AFND pedido ya reenumerados sus estados es:





¡ Solución !

Los componentes del AFND para la expresión regular $NumEsp \Rightarrow Dig^{+}.Dig^{+}$ son los que a continuación se mencionan.

s_0 = 0
S = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
Σ = { 0, 1, ... , 9,**.** }
F = { 9 }

Función *move (s,a) :*

Símbolos en la entrada

Estados	**Dig**	**.**	**∈**
0	{ 1 }	-	-
1	-	-	{ 2,4 }
2	{ 3 }	-	-
3	-	-	{ 2,4 }
4	-	{ 5 }	-
5	{ 6 }	-	-
6	-	-	{ 7,9 }
7	{ 8 }	-	-
8	-	-	{ 7,9 }

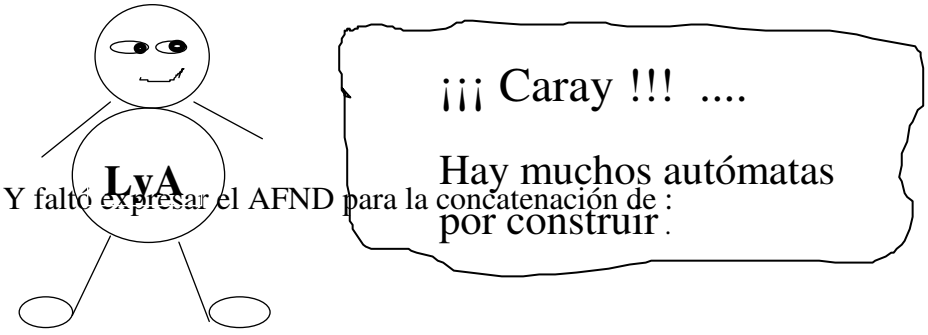
move (0, .) = Error
move (0, ∈) = Error
move (1, Dig) = Error
move (1, .) = Error
move (2, .) = Error
move (2, ∈) = Error

Ejemplo 3.7 *La definición regular para el token* ***num*** *en Pascal es :*

Dig ⇒ *0 | 1 | ... | 9*
FraccionOpcional ⇒ *(. Dig$^+$) ?*
ExponenteOpcional ⇒ *(E (+ | -) ? Dig$^+$) ?*
Num ⇒ *(Dig$^+$) (FraccionOpcional) (ExponenteOpcional)*

Construir el AFND que reconozca el lenguaje denotado por la expresión regular Num.
La descomposición sintáctica :

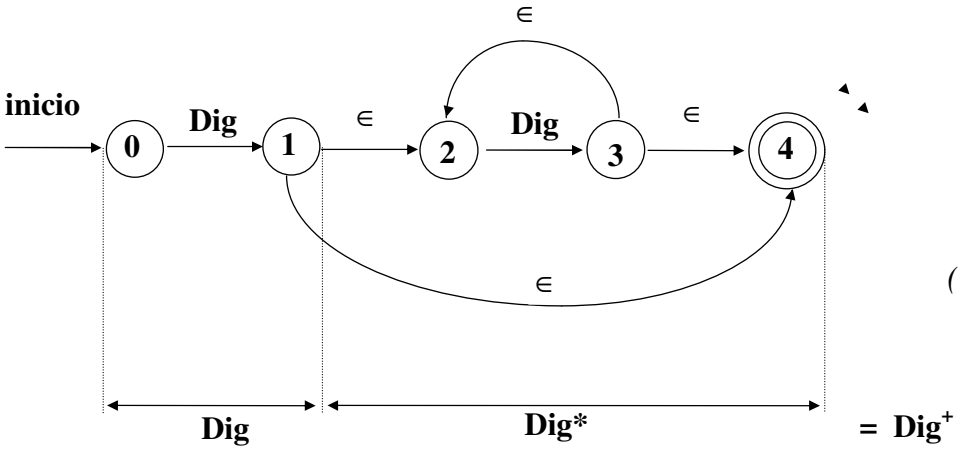
Dig$^+$	Dig
	Dig*
	DigDig* // Dig$^+$
FraccionOpcional	.
	Dig
	Dig*
	Dig$^+$ // DigDig*
	. Dig$^+$ // .DigDig*
	(. Dig$^+$) ?
ExponenteOpcional	E
	+
	-
	+ | -
	(+ | -) ?
	E (+ | -) ?
	Dig
	Dig*
	DigDig* // Dig$^+$
	E (+ | -) ? Dig$^+$
	(E (+ | -) ? Dig$^+$) ?



Y faltó expresar el AFND para la concatenación de :

(Dig$^+$) (FracciónOpcional) (ExponenteOpcional) .

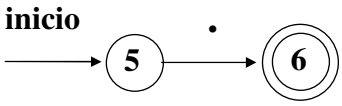
Dig$^+$ es una expresión regular ya conocida por nosotros. Su AFND es obtenido tras de aplicar las reglas 2, 3 c) y 3 b) :



(3.7.1)

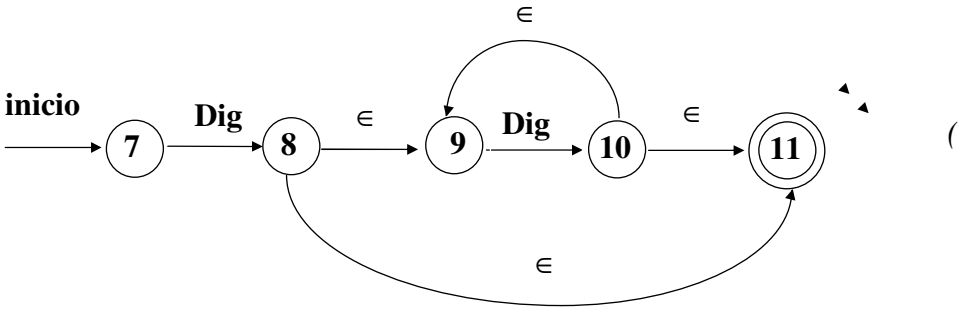
Sigamos con la concatenación **.***Dig*$^+$ perteneciente a la expresión regular *FracciónOpcional :*

AFND para el símbolo "**.**"



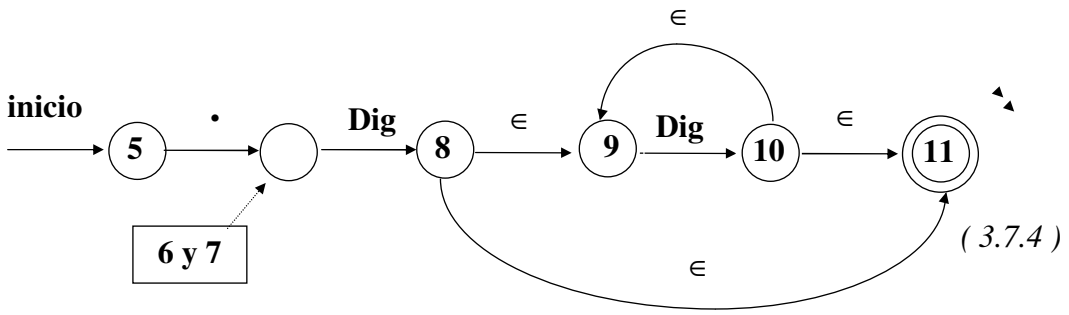
(3.7.2)

AFND para *Dig*$^+$:

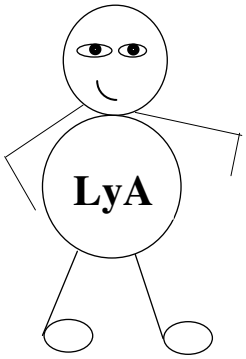


(3.7.3)

Efectuamos la concatenación de (3.7.2) y (3.7.3), mezclando los estados 6 y 7 en un solo estado:



(3.7.4)



Pero .....

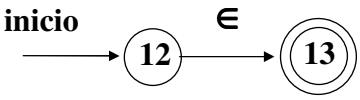
Qué regla aplicamos para obtener :

$(. \text{Dig}^{+}) ?$

Sabemos que no hay regla para *r ?,* pero también sabemos que :

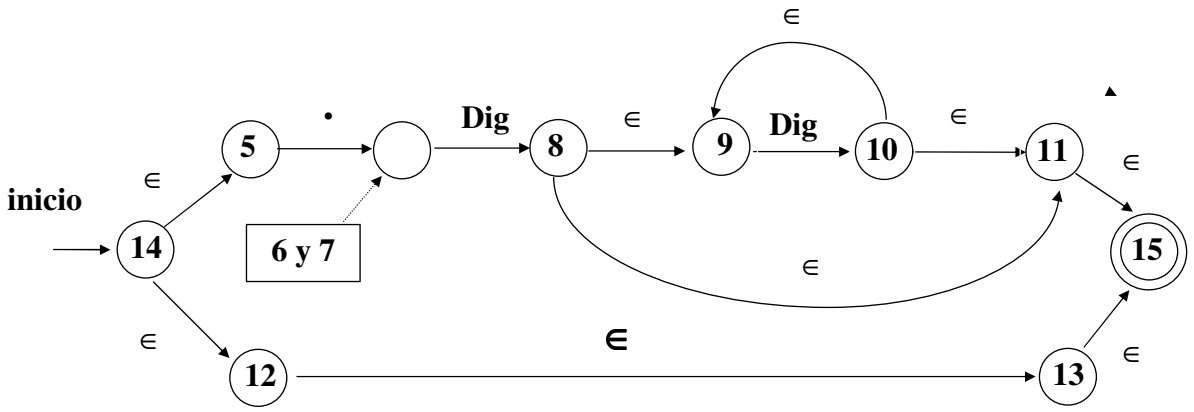
r ? = r | ∈

Así $(. Dig^{+}) ?$ = $. Dig^{+}$ | ∈. Tenemos el AFND para el primer término de la alternancia, falta obtener el AFND para el símbolo ∈.



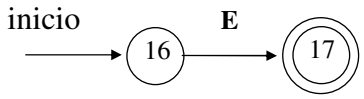
(3.7.5)

Y con la regla 3 a, unimos 3.7.4 con 3.7.5, y tenemos el AFND para la alternancia

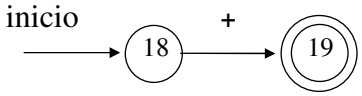


(3.7.6) AFND para *(.Dig*$^{+}$ *) ?* expresión regular *FraccionOpcional.*

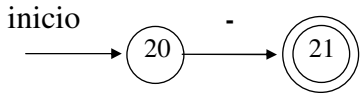
Ya tenemos los AFND's para dos de los tres términos de la concatenación que define a la expresión regular *Num*. Construyamos el AFND para la subexpresión regular *ExponenteOpcional.*
Pasos en la construcción del AFND para *E (+ | -) ?* :



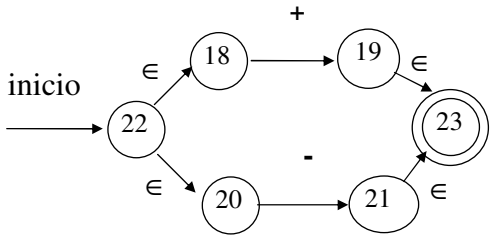
// AFND símbolo 'E' ; regla 2.



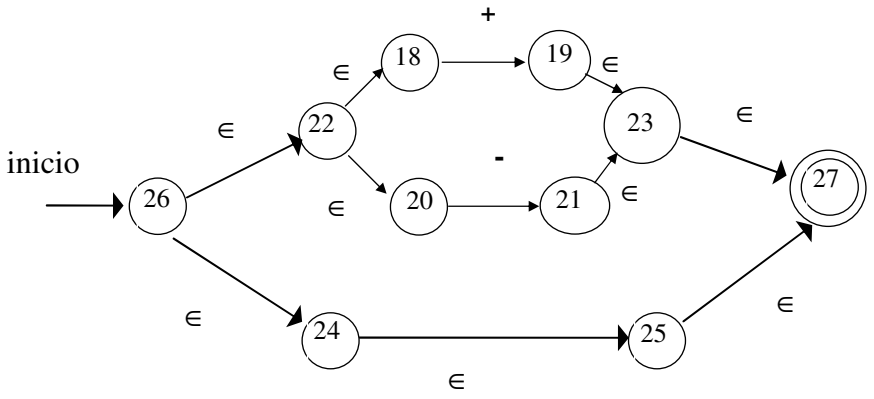
// AFND símbolo '+' ; regla 2.



// AFND símbolo '-' ; regla 2.

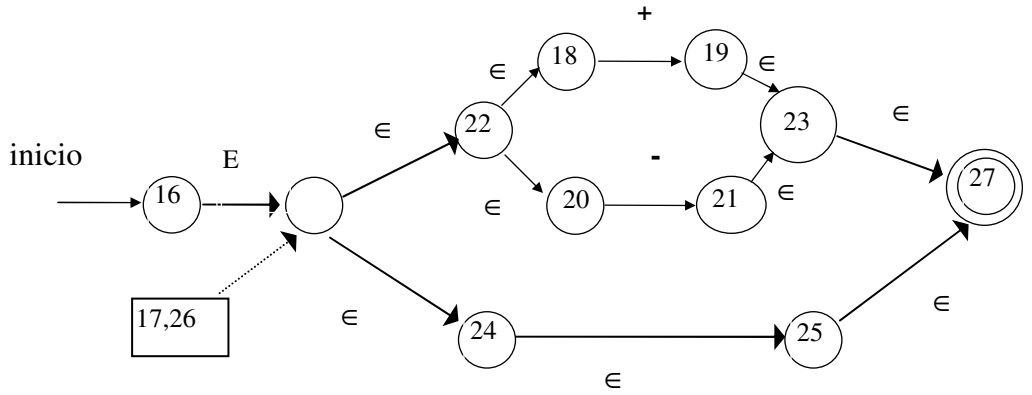


// AFND para + | - ; regla 3 a).



AFND para (+ | -) ? ; regla 3 a)

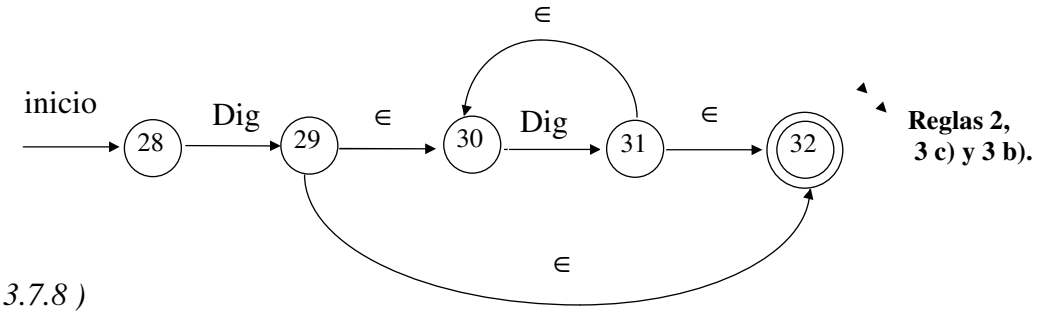
(+ | -) ? = (+ | -) | ∈



(3.7.7)

AFND para *E (+ | -) ?,* la concatenación produce la mezcla de los estados 17 y 26 en un sólo estado.

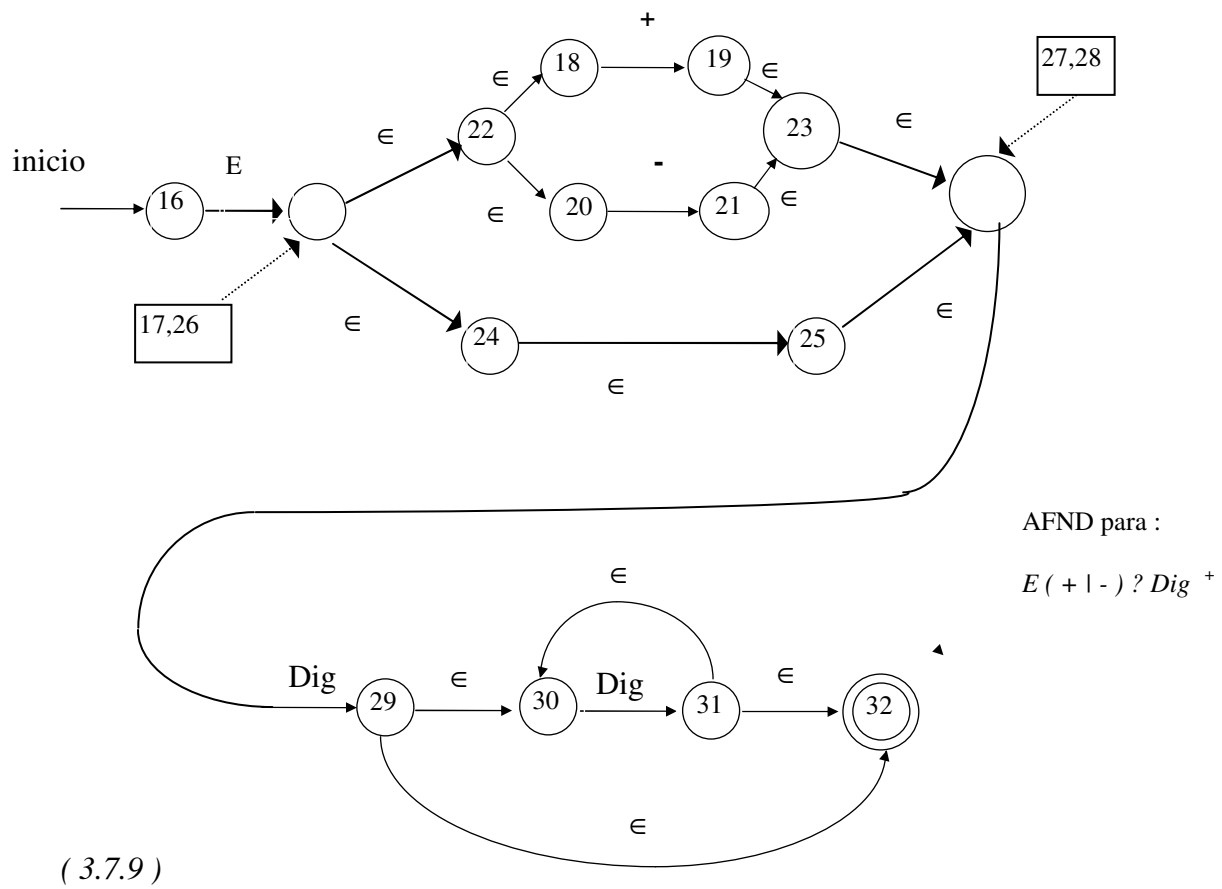
Ahora, construyamos el AFND para *E (+ | -) ? Dig* $^{+}$, iniciando por obtener *Dig* $^{+}$:



Reglas 2, 3 c) y 3 b).

(3.7.8)

La aplicación de la regla 3 b) para la concatenación de (3.7.7) y (3.7.8) obliga a la mezcla de los estados 27 y 28 en uno solo :

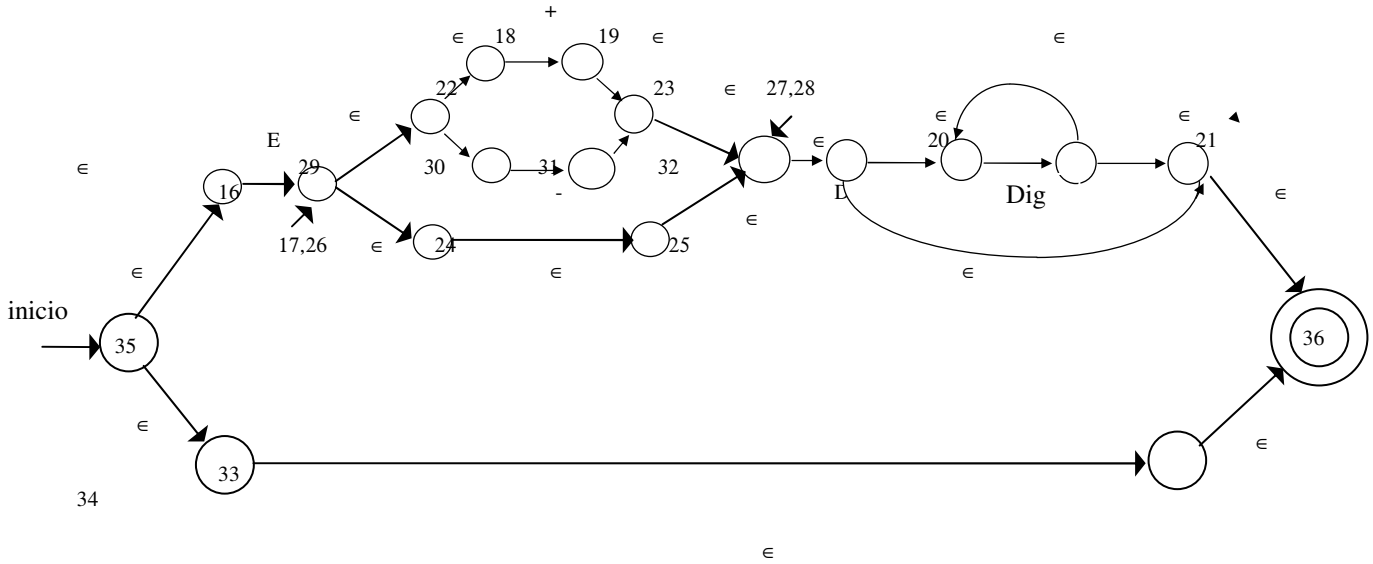


(3.7.9)

El AFND que reconoce :

$$(E (+ | -) ?\ Dig^{+}) ? \ = \ E (+ | -) ?\ Dig^{+} \ | \ \in$$

es encontrado aplicando la regla 1 y 3 a), para el autómata (3.7.9) y el autómata para el símbolo ∈ :



(3.7.10) AFND para la expresión regular *ExponenteOpcional*

El AFND que reconoce el token *Num,* consiste de la concatenación de los AFND's :

3.7.1 *Dig* $^{+}$,
3.7.6 *FraccionOpcional* y
3.7.10 *ExponenteOpcional*.

La Fig. 3.11 muestra a dicho autómata.

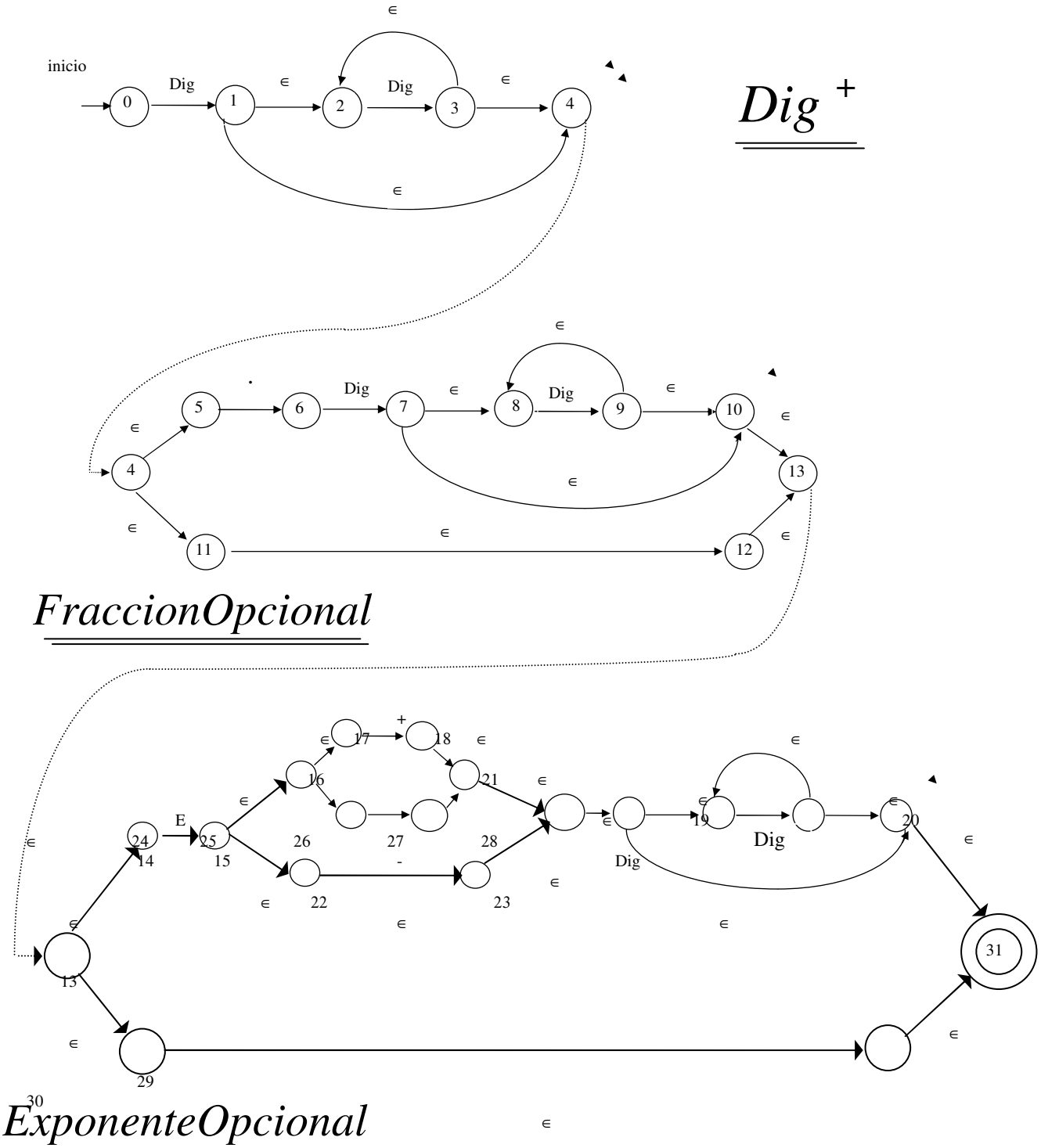


Fig. 3.11 AFND para el token *Num*, con los estados reenumerados.

3.5 EQUIVALENCIA ENTRE UN AFND Y UN AFD.

La naturaleza de un *autómata finito no determinístico*, hace que la tarea de simularlo en un programa de computadora sea muy difícil. Para un símbolo de entrada pueden existir diferentes transiciones estado a estado, es decir, la aceptación de una cadena puede o debe, involucrar diferentes trayectorias y todas ellas deben de probarse para saber si la cadena de entrada *es o no* , reconocida por el AFND.
Dado que las reglas y algoritmo de construcción de Thompson son susceptibles de ser programados en una computadora, sin que ésto sea de gran complejidad, una buena estrategia es obtener el AFND que reconozca el lenguaje denotado por *una expresión regular*, para luego construir un *AFD* a partir de un AFND. El algoritmo que construye un *AFD* dado un AFND, se denomina **Construcción de subgrupos**. Una vez encontrado el autómata finito determinístico, puede ser optimizado o sea, reducir el número de estados **S** que lo forman.

El algoritmo de *construcción de subgrupos* tiene como principal tarea, construir una tabla de transición (función *move*) para el nuevo AFD. Cada estado del AFD es un conjunto de estados del AFND. Además, el algoritmo hace uso de las operaciones que aparecen en la tabla de la fig. 3.12.

Operación	Descripción
$\in$*–cerradura (s)*	Es el conjunto de estados del AFND que son "alcanzados" desde el estado **s** del AFND, con transiciones (arcos) $\in$ solamente.
$\in$*-cerradura (T)*	Es el conjunto de estados del AFND que son "alcanzados" desde algún estado **s** en **T** del AFND, con arcos $\in$ (transiciones) solamente.
move (T,a)	Es el conjunto de estados del AFND hacia los cuales hay una transición con un símbolo **a** en la entrada, desde algún estado **s** en **T** del AFND.

Fig. 3.12 Operaciones de soporte para obtener un AFD.

Tanto el algoritmo de *construcción de subgrupos*, como el algoritmo para el cálculo de la operación auxiliar *∈-cerradura(T),* se muestra en la fig. 3.13 a) y b).

Entrada

AFND Autómata finito no determinístico.

Proceso

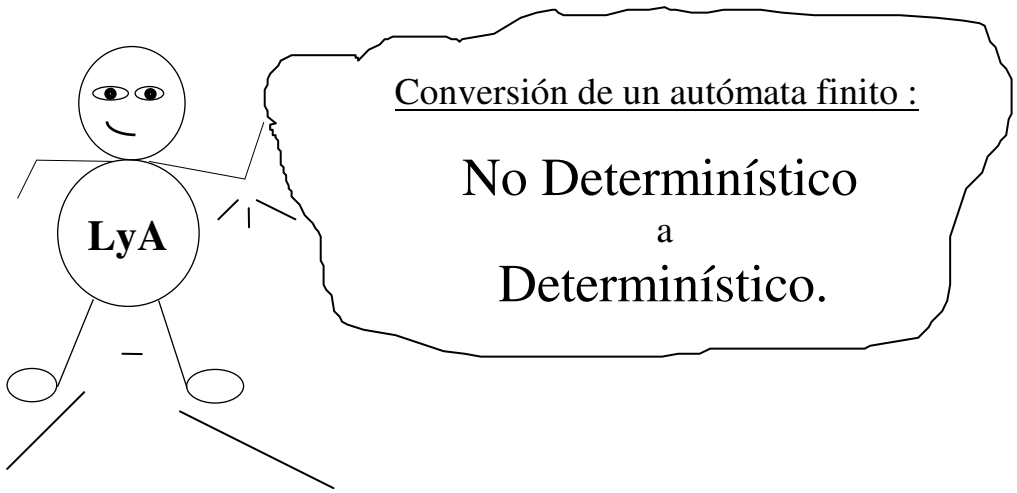
D = ∈-cerradura (s_o) *D* es el conjunto de estados del nuevo AFD. Inicialmente los estados calculados en esta cerradura decimos que **no están marcados.**

```
While  ( existe un estado T no marcado en D ) Do
   Begin
      Marcar T
      For ( cada símbolo de entrada a ) Do
         Begin
            U =  ∈-cerradura (move(T,a))
            if  ( U no está en D )
               then
                  añadir U como un estado no marcado a D
            D_tran [T,a] = U
         end
   end
```

Salida

D_{tran} : Tabla de transición del AFD.

Fig. 3.13 (a) Algoritmo de construcción de subgrupos.



Entrada

T : Cualesquier conjunto de estados del AFND.

Proceso

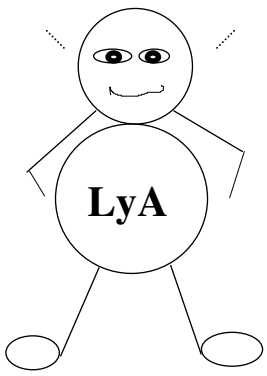
```
Meter todos los estados en T a la pila
∈−cerradura (T) = T
While ( Pila no está vacia ) Do
   Begin
      Sacar t, elemento tope de la pila
      for ( cada estado u con un arco ∈ desde t a u ) Do
        if ( u no está en ∈−cerradura (T) )
          then
            Begin
              añadir u a ∈−cerradura (T)
              meter u a la pila
            end
   end
```

Salida

∈-cerradura (T)

Fig. 3.13 (b) Cálculo de la *∈-cerradura.*

Ejemplo 3.8. *Utiliza el algoritmo de construcción de subgrupos, para construir el AFD equivalente al AFND (3.4.1), obtenido en el ejemplo 3.4.*



¿ Cómo empiezo ?

Los algoritmos se ven muy feos !!!!

Bueno, lo primero es dibujar la tabla de transición denominada D_{Tran}, la cual es la salida del algoritmo de construcción de subgrupos Fig. 3.14.

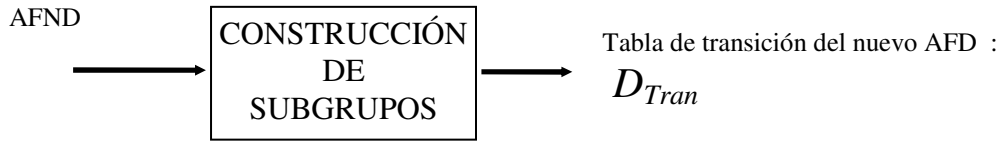


Fig. 3.14. Entradas y salidas para el algoritmo de *construcción de subgrupos.*

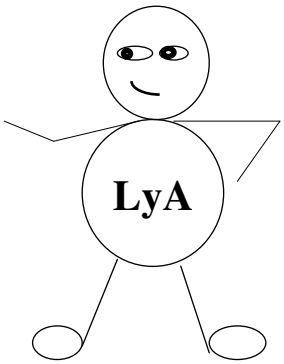
D_{Tran}

Símbolos en la entrada

Estados		**c**	**d**	**a**

Ahh !! , las <u>columnas</u> en la tabla son las posibles entradas (símbolos del alfabeto Σ), y los <u>renglones</u> son los estados.

¿ Cuáles estados ? !!!



Los renglones de la tabla D_{tran} son los estados del nuevo AFD que vamos a obtener, aplicando el algoritmo de *construcción de subgrupos*. Debemos tener cuidado, tal y como se establece en la sección 3.1, que los símbolos de entrada c_1 , c_2 ,.. c_k cumplan :

$c_i \cap c_j = \emptyset$, para $i \neq j$

En nuestro caso $c_1 = c$, $c_2 = d$, $c_3 = a$:

$c \cap d = \emptyset$
$c \cap a = \emptyset$
$d \cap a = \emptyset$

¡¡¡ Si se cumple !!!

Si lo anterior no se cumple, corremos el peligro de construir un AFD que **no sea determinístico!** o sea, un *AFND!,* lo cual sería totalmente erróneo ya que precisamente el algoritmo lo usamos para formar un <u>*AFD*</u> a partir de un AFND.

Además $\bigcup_{i=1}^{n} c_i = \Sigma$ alfabeto de entrada del nuevo AFD menos el símbolo $\in$, lo cual se cumple :

$$\Sigma = \{ a , c , d \}$$

Iniciemos con la primera instrucción del algoritmo :
$\in$*-cerradura* (s_0) = $\in$*-cerradura* (*s{0}*)

La $\in$*-cerradura (s{0})* del estado de inicio nos permite obtener el primer estado del nuevo autómata AFD. Llamémosle *A* y lo añadimos a *D* (conjunto de estados del nuevo AFD).

$$A = \in\text{-}cerradura\ (s\{0\}) \qquad (3.8.1)$$

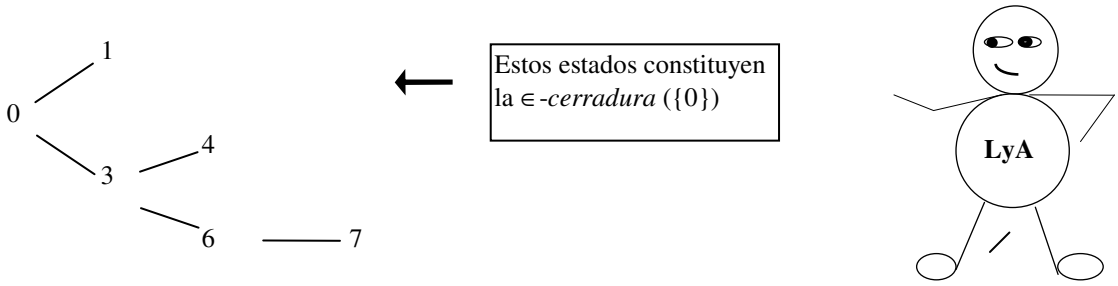
Apliquemos el algoritmo de la fig. 3.13 b) para el cálculo de la $\in$*-cerradura* , donde :

T = {0}

La $\in$*−cerradura* ({0}) se inicializa a *T* :

$$\in\text{-}cerradura\ (\{0\}) = \{0\}$$

Realmente, el algoritmo de la Fig. 3.13 b), usa una pila para almacenar información de estados que "no han sido probados", es decir, los estados en la pila tienen que ser visualizados para ver si **<u>alcanzan</u>** con arcos $\in$ algún otro u otros estados. Si es así, estos nuevos estados "alcanzados" son metidos a la pila y añadidos a la $\in$*−cerradura (T).* Asimismo estos estados son también "probados" para ver si ellos "alcanzan" a otros estados, y así recurrentemente se repite el proceso, hasta que la pila quede vacía.
Para este ejemplo, observemos el AFND 3.4.1. y mostremos gráficamente a los estados que son aceptados con los arcos $\in$ desde el estado de inicio $s_0 = 0$:



Así, $\in$*−cerradura* ({0}) = { 0,1,3,4,6,7 } por lo que el primer estado del nuevo autómata AFD es en realidad el conjunto de estados { 0,1,3,4,6,7 } del AFND :

A = { 0,1,3,4,6,7 }

Agreguemos este estado así calculado a la tabla de transición D_{tran}.

Símbolos en la entrada

Estados
{0,1,3,4,6,7}

	c	**d**	**a**
A			

Ahora el conjunto de estados del nuevo AFD *D ={A},* tiene un nuevo estado, el primero!

Enseguida, el algoritmo de la *construcción de subgrupos*, establece que hay que obtener la transición que da lugar cuando el AFD se encuentra en el estado *A* y en la entrada se tiene el símbolo *c*. Lo mismo se calcula para el símbolo *d* y el símbolo *a*. Estas transiciones son los nuevos estados para el AFD que se deberán añadir al conjunto *D*.

Los nuevos estados son calculados mediante la $\in$*-cerradura* de las funciones *move* para el estado *A* y cada símbolo en la entrada.

B = $\in$*−cerradura (move(A,c))*
C = $\in$*−cerradura (move(A,d))*
D = $\in$*−cerradura (move(A,a))*
Calculemos primeramente a todos los *move's.*

El *move (A,c)* es el conjunto de estados del AFND que son alcanzados desde *A*, al existir en la entrada un símbolo *c*. Recordemos que *A* = { 0,1,3,4,6,7 }. Entonces :

move ({0,1,3,4,6,7}, c) = {2}

ya que el estado 2 es alcanzado desde el estado 1 con una entrada *c*, (ver autómata 3.4.1), y el estado 1 está en *A*.

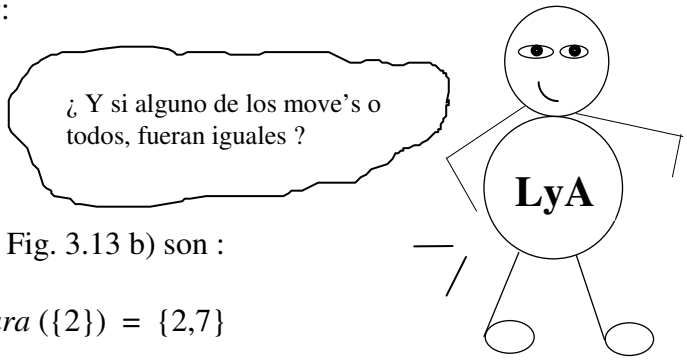
move ({0,1,3,4,6,7}, d) = {5}

Debido a que el estado 5 es alcanzado desde el estado 4, y el estado 4 está en *A*.

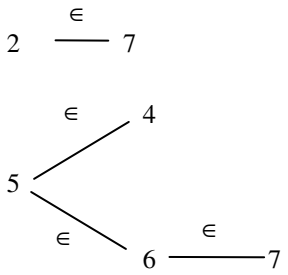
move ({0,1,3,4,6,7}, a) = {8}

En este caso el estado 7 "alcanza" al estado 8, con un arco etiquetado por <u>*a*</u>. Ahora nos resta obtener las cerraduras sobre estos *move's*:

$B = \in\text{-}cerradura\ (\{2\})$
$C = \in\text{-}cerradura\ (\{5\})$
$D = \in\text{-}cerradura\ (\{8\})$



Las $\in$-*cerraduras* aplicando el algoritmo de la Fig. 3.13 b) son :



$B = \in\text{-}cerradura\ (\{2\}) = \{2,7\}$

$C = \in\text{-}cerradura\ (\{5\}) = \{5,4,6,7\}$

8 $\qquad D = \in\text{-}cerradura\ (\{8\}) = \{8\}$

<u>*B, C y D* nuevos estados del AFD.</u>

Se añaden estos nuevos estados a D_{estados} :

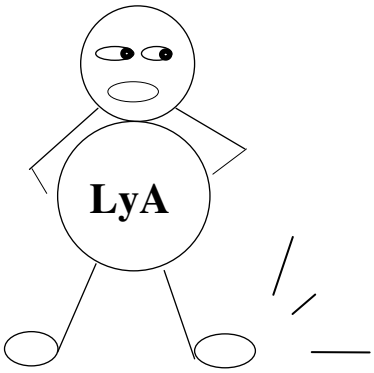
$D_{estados}$ = { A,B,C,D }

Y también a la tabla de transición, como renglones. Recordemos que *B, C, D* son estados a los cuales hay una transición desde el estado *A*, habiendo un símbolo en la entrada.
La tabla de transición D_{Tran} tiene ahora nuevos elementos:

Estados	**Símbolos en la entrada**			
		c	**d**	**a**
{0,1,3,4,6,7}	A	B	C	D
{2,7}	B			
{4,5,6,7}	C			
{8}	D			

El algoritmo de *construcción de subgrupos* dice que el estado *A* "ya está marcado", es decir, las transiciones (si existen) del estado *A* para cada símbolo en la entrada, ya se encontraron. Ahora los estados *B, C y D* son los que aún no están marcados, o sea que tenemos que encontrar las transiciones del estado *B*, del *C* y del *D*, para cada símbolo en la entrada.
De acuerdo a lo anterior, calculemos en primer término las $\in$*−cerraduras* de los *move's* del estado $B = \{2,7\}$.

$E = \in$*−cerradura* (*move*(B,c))
$F = \in$*−cerradura* (*move*(B,d))
$G = \in$*−cerradura* (*move*(B,a))



Estas $\in$*−cerraduras* generan 3 nuevos estados *si y sólo si,* las $\in$*−cerraduras* no son vacías.

Es decir, sólo la $\in$*−cerradura* que no sea $= \emptyset$ genera un nuevo estado en la tabla D_{tran} del AFD que se está construyendo.

Obtengamos primero los *move's*, tal y como lo hicimos anteriormente con el estado *A :*

move (B,c) = *move* ({2,7}, c) = Ø // *vacío*
move (B,d) = *move* ({2,7}, d) = Ø // *vacío*
move (B,a) = *move* ({2,7}, a) = {8}

Luego :

$E = \in$*−cerradura* (vacío) = No existe transición, por lo tanto, tampoco el estado *E*.
$F = \in$*−cerradura* (vacío) = No existe transición, por lo tanto, tampoco el estado *F*.
$G = \in$*−cerradura* ({8}) = {8}, ya que desde el estado 8 no se alcanza ningún estado con un arco $\in$.

Además, $G = D$. Y la tabla de transición con estos cambios queda :

Estados	Símbolos en la entrada		
	c	**d**	**a**
{0,1,3,4,6,7} A	B	C	D
{2,7} B	-	-	D
{4,5,6,7} C			
{8} D			

Seguimos con el estado *C*.

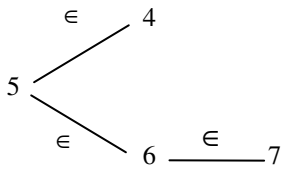
$\in-$*cerradura* (*move*(C,c))
$\in-$*cerradura* (*move*(C,d))
$\in-$*cerradura* (*move*(C,a))

Sus funciones *move* son :

move (C,c) = *move* ({4,5,6,7}, c) = Ø // vacío
move (C,d) = *move* ({4,5,6,7}, d) = {5}
move (C,a) = *move* ({4,5,6,7}, a) = {8}

y las cerraduras son :

$\in-$*cerradura* (vacío) = No existe, no hay transición.
$\in-$*cerradura* ({5}) = {4,5,6,7}, es el estado *C*.



$\in-$*cerradura* ({8}) = {8}, es el estado *D*.

Agregamos estas transiciones a la tabla de transición D_{Tran}

Estados	Símbolos en la entrada		
	c	**d**	**a**
{0,1,3,4,6,7} A	B	C	D
{2,7} B	-	-	D
{4,5,6,7} C	-	C	D
{8} D			

Lo mismo hacemos para el estado D.

$\in-$*cerradura* (*move*(D,c))
$\in-$*cerradura* (*move*(D,d))
$\in-$*cerradura* (*move*(D,a))

Los *move's* son:

move (D,c) = *move* ({8}, c) = Ø // vacío
move (D,d) = *move* ({8}, d) = Ø // vacío
move (D,a) = *move* ({8}, a) = Ø // vacío

y las cerraduras también son vacías, es decir, del estado *D* a otro estado <u>no existen transiciones.</u>

	Símbolos en la entrada			
Estados		**c**	**d**	**a**
{0,1,3,4,6,7}	A	B	C	D
{2,7}	B	-	-	D
{4,5,6,7}	C	-	C	D
{8}	D	-	-	-

El estado *D* puede eliminarse de la tabla de transición dado que no tiene transiciones a otro estado.

	Símbolos en la entrada			
Estados		**c**	**d**	**a**
{0,1,3,4,6,7}	A	B	C	D
{2,7}	B	-	-	D
{4,5,6,7}	C	-	C	D

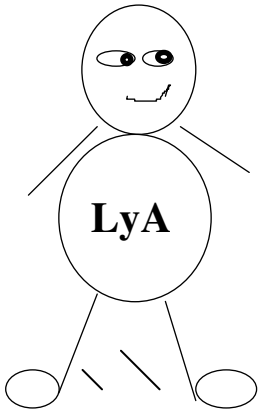
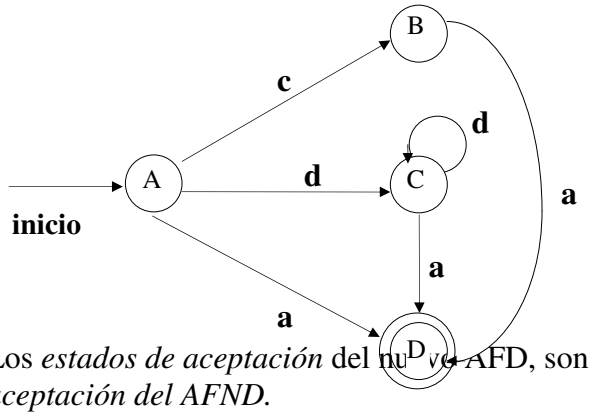
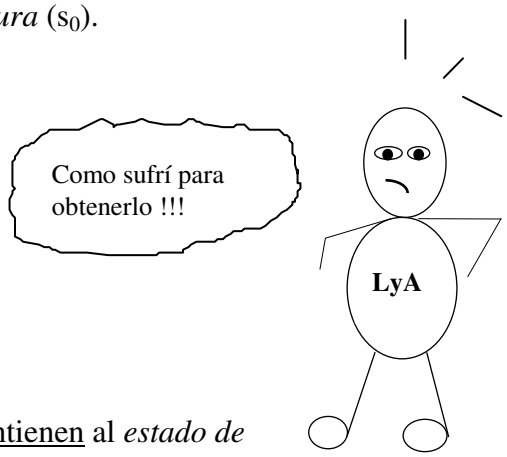


Tabla de transición D_{tran} del autómata finito determinístico AFD, resultado de aplicar el algoritmo de construcción de subgrupos.

Con la tabla D_{Tran} así obtenida, podemos mostrar el diagrama del AFD. El estado de inicio del nuevo AFD siempre será *A*, es decir la $\in$*−cerradura* (s_0).





Los *estados de aceptación* del nuevo AFD, son los que <u>contienen</u> al *estado de aceptación del AFND.*

A = {0,1,3,4,6,7}	//	No contiene al estado 8.
B = {2,7}	//	No contiene al estado 8.
C = {4,5,6,7}	//	No contiene al estado 8.
D = {8}	//	**<u>Si lo contiene.</u>**

Por lo tanto, *D* es el estado (en este caso, el único) de *aceptación* del nuevo **AFD.**

Ejemplo 3.9. *Dado el autómata finito no determinístico AFND (3.5.8) del ejemplo 3.5. obtener su correspondiente AFD aplicando el algoritmo de construcción de subgrupos.*

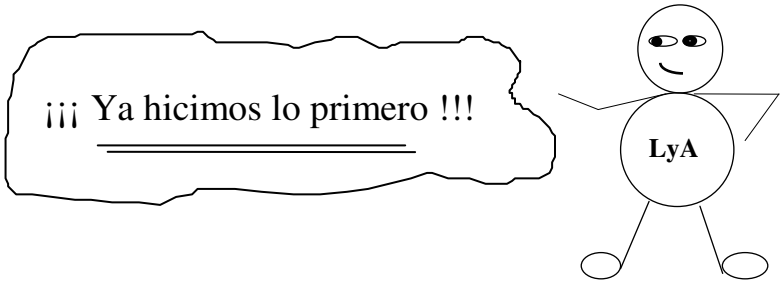
Iniciamos con las columnas por la tabla D_{tran}.

Símbolos en la entrada

Estados		**Letra**	**Dig**	**Sub**
	.	.	.	.

Es fácil ver que :

Letra $\cap$ *Dig* = Ø
Letra $\cap$ *Sub* = Ø
Dig $\cap$ *Sub* = Ø



Ahora, tenemos que encontrar los estados que forman al conjunto *D* y añadirlos junto a sus transiciones, a la tabla de transición D_{tran}.

El primer estado del nuevo AFD (estado de inicio) es :

A $= \in\text{-}cerradura\ (s_0)$

donde s_0 *es el estado de inicio del AFND.*

La $\in$*-cerradura* ({0}) se obtiene aplicando el algoritmo de la Fig. 3.13 b).

$$A = \in\text{-}cerradura\ (\{0\}) = \{0\}$$

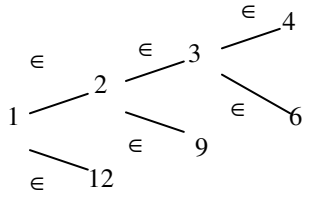
Calculamos las transiciones del estado *A*, cuando en la entrada se tiene una *letra* o un *dígito* o bien un *subrayado.*

$\in$*-cerradura* (*move* (A,*letra*))
$\in$*-cerradura* (*move* (A,*Dig*))
$\in$*-cerradura* (*move* (A,*Sub*))
Sus funciones *move* son :

move (A,letra) = *move* ({0},*letra*) = {1}
move (A,Dig) = *move* ({0},*Dig*) = Ø // vacío
move (A,Sub) = *move* ({0},*Sub*) = Ø // vacío

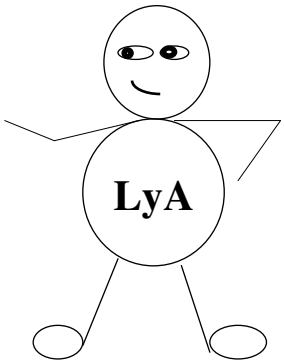
Sólo la $\in$*-cerradura* (*move*(A,l*etra*)) es diferente al conjunto vacío. Vamos a obtenerla :

$$B = \in\text{-}cerradura\ (\{1\}) = \{\ 1,2,3,4,6,9,12\ \}$$



Esta $\in$*-cerradura* constituye al nuevo estado del AFD :

B = {1,2,3,4,6,9,12 }

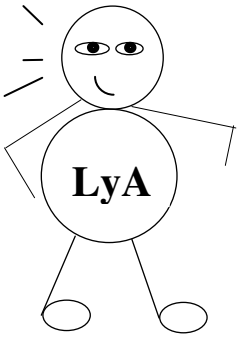


La tabla de transición D_{tran} después de añadir los estados *A* y *B* y las transiciones de *A*, se muestran enseguida.

Símbolos en la entrada

Estados		**Letra**	**Dig**	**Sub**
{0}	A	B	-	-
{1,2,3,4,6,9,12}	B			
	.	.	.	.

El estado *A*, se dice que está "marcado", ya que sus transiciones para los símbolos de entrada fueron calculados. El estado *B* no está "marcado", pues falta calcular las transiciones, que posiblemente nos lleven a <u>*nuevos estados*</u>.

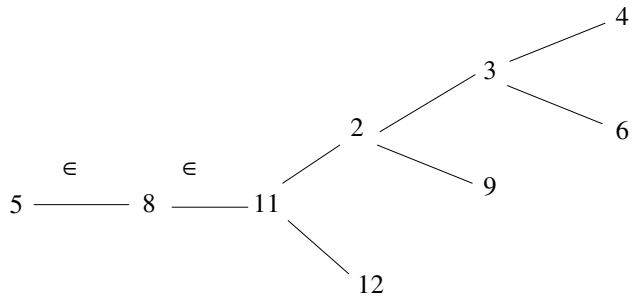


Pues ... calculemos las transiciones para el estado B.

- $\in-$*cerradura* (*move*(B,Letra))

  move({1,2,3,4,6,9,12} , Letra) = {5} // El estado 4 alcanza al 5. Ver AFND 3.5.8

$\in-$*cerradura* ({5}) = {5,8,11,2,3,4,9,6,12} // <u>Nuevo estado</u>

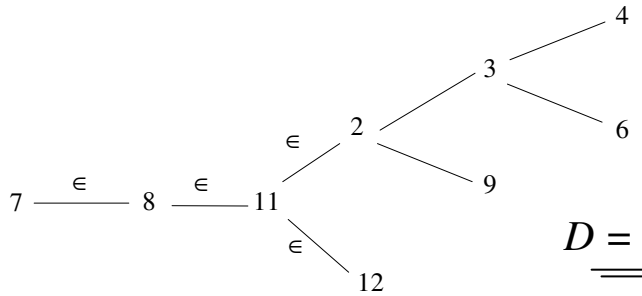


Llamémosle *C* al nuevo estado del AFD, $C = \{2,3,4,5,6,8,9,11,12\}$

- $\in-$*cerradura* (*move*(B,Dig))

  move ({1,2,3,4,6,9,12},Dig) = {7} // El estado 6 alcanza al 7. Ver AFND 3.5.8

$\in-$*cerradura* ({7}) = {7,8,11,2,3,4,6,9,12} // <u>Nuevo estado</u>

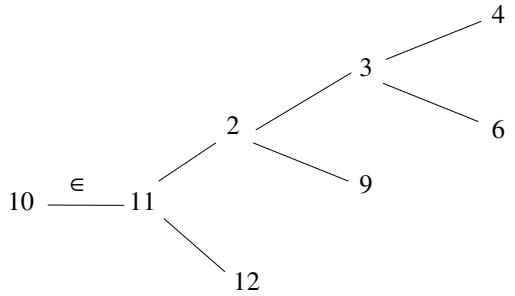


$D = \{2,3,4,6,7,8,9,11,12\}$

- $\in-$*cerradura* (*move*(B,Sub))

  move({1,2,3,4,6,9,12},Sub) = {10} // El estado 9 alcanza al 10. Ver AFND 3.5.8

$\in-$*cerradura* ({10}) = {10,11,2,3,4,6,9,12} // Nuevo estado



$E = \{2,3,4,6,9,11,12\}$

Ahora, el AFD tiene ya 5 estados : $D_{Estados} = \{ A,B,C,D,E \}$. Sumemos estos nuevos estados a D_{tran} , además de las transiciones de *B* con cada símbolo en la entrada.

Estados	Símbolos en la entrada			
		Letra	**Dig**	**Sub**
{0}	A	B	-	-
{1,2,3,4,6,9,12}	B	C	D	E
{2,3,4,5,6,8,9,11,12}	C			
{2,3,4,6,7,8,9,11,12}	D			
{2,3,4,6,9,11,12}	E			

Los estados *A* y *B* ya están marcados. Los estados *C*, *D* y *E* no lo están, de acuerdo al algoritmo de *construcción de subgrupos*. Esto quiere decir, que es necesario calcular las transiciones de los estados *C*, *D* y *E* para cada símbolo en la entrada.

Transiciones del estado *C*.

- $\in-$*cerradura* (*move*(C,Letra))

move ({2,3,4,5,6,8,9,11,12},Letra) = {5}
$\in$*−cerradura* ({5}) = {5,8,11,2,3,4,9,6,12} // Estado *C*

- $\in$*−cerradura* (*move*(C,Dig))
  move ({2,3,4,5,6,8,9,11,12},Dig) = {7} // Ya calculado. Este *move* nos lleva al estado *D*.

$\in$*−cerradura* ({7}) = Estado *D*

- $\in$*−cerradura* (*move*(C,Sub))
  move ({2,3,4,5,6,8,9,11,12},Sub) = {10} // Ya calculado. Este *move* nos lleva al estado *E*.

$\in$*−cerradura* ({10}) = Estado *E*

La tabla D_{tran} con las anteriores transiciones ya incluídas es :

Estados	Símbolos en la entrada			
		Letra	**Dig**	**Sub**
{0}	A	B	-	-
{1,2,3,4,6,9,12}	B	C	D	E
{2,3,4,5,6,8,9,11,12}	C	C	D	E
{2,3,4,6,7,8,9,11,12}	D			
{2,3,4,6,9,11,12}	E			

Transiciones estado *D*.

- $\in$*−cerradura* (*move*(D,Letra))
  move ({2,3,4,6,7,8,9,11,12},Letra) = {5} // Ya calculado. Este *move* nos lleva al estado *C*.

$\in$*−cerradura* ({5}) = Estado *C*

- $\in$*−cerradura* (*move*(D,Dig))
  move ({2,3,4,6,7,8,9,11,12},Dig) = {7} // Ya calculado. Este *move* nos lleva al estado *D*.

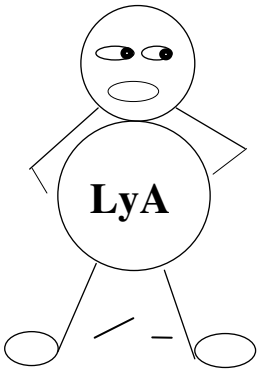
$\in$*−cerradura* ({7}) = Estado *D*

- $\in$*−cerradura* (*move*(D,Sub))
  move ({2,3,4,6,7,8,9,11,12},Sub) = {10} // Ya calculado. Este *move* nos lleva al estado *E*.

$\in$*−cerradura* ({10}) = Estado *E*

Incluímos estas transiciones del estado *D* en la tabla D_{tran} :

Estados	Símbolos en la entrada			
		Letra	**Dig**	**Sub**
{0}	A	B	-	-
{1,2,3,4,6,9,12}	B	C	D	E
{2,3,4,5,6,8,9,11,12}	C	C	D	E
{2,3,4,6,7,8,9,11,12}	D	C	D	E
{2,3,4,6,9,11,12}	E			



Sólo falta obtener las transiciones para el estado E.

<u>Transiciones estado *E*.</u>

- $\in$*−cerradura* (*move*(E,Letra))
  move ({2,3,4,6,7,8,9,11,12},Letra) = {5} // Ya calculado. Este *move* nos lleva al estado *C*.

$\in$*−cerradura* ({5}) = <u>Estado *C*</u>

- $\in$*−cerradura* (*move*(E,Dig))
  move ({2,3,4,6,7,8,9,11,12},Dig) = {7} // Ya calculado. Este *move* nos lleva al estado *D*.

$\in$*−cerradura* ({7}) = <u>Estado *D*</u>

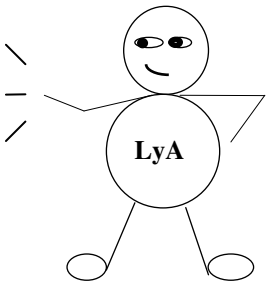
- $\in$*−cerradura* (*move*(E,Sub))
  move({2,3,4,6,7,8,9,11,12},Sub) = {10} // Ya calculado. Este *move* nos lleva al estado *E*.

$\in$*−cerradura* ({10}) = <u>Estado *E*</u>

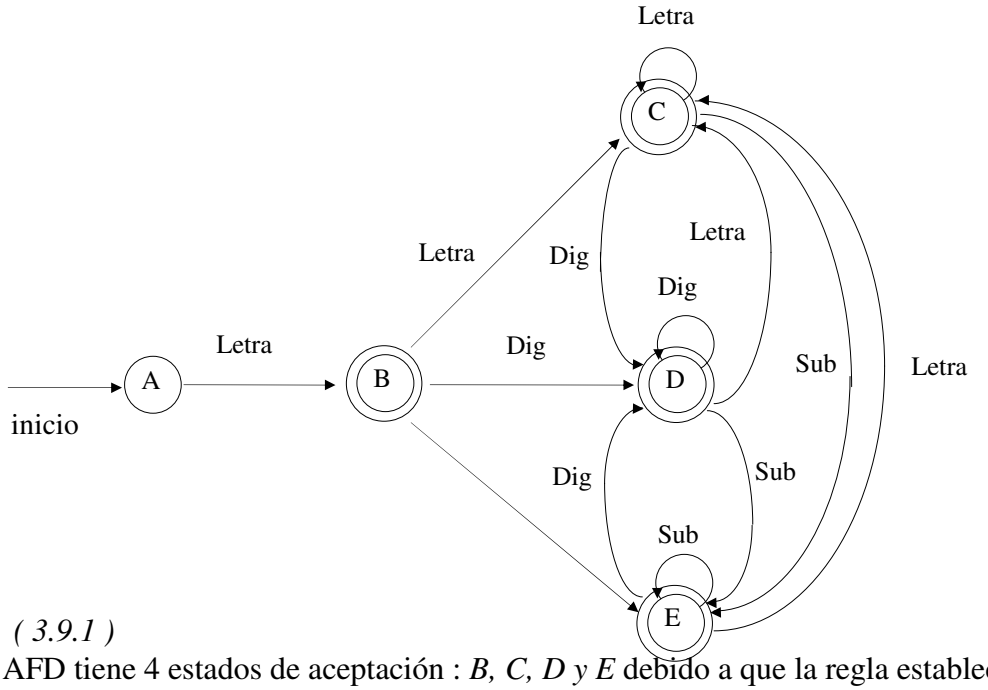
La tabla D_{tran} del nuevo AFD queda :

Estados		Símbolos en la entrada		
		Letra	**Dig**	**Sub**
{0}	A	B	-	-
{1,2,3,4,6,9,12}	B	C	D	E
{2,3,4,5,6,8,9,11,12}	C	C	D	E
{2,3,4,6,7,8,9,11,12}	D	C	D	E
{2,3,4,6,9,11,12}	E	C	D	E

Todos los estados *A, B, C, D y E* están "marcados", es decir, sus transiciones han sido calculadas. El algoritmo de *construcción de subgrupos* ha terminado.



El diagrama del nuevo autómata se construye a partir de la tabla de transición D_{tran}.

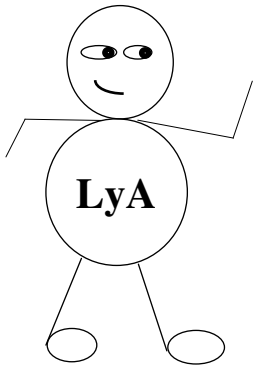


(3.9.1)

El AFD tiene 4 estados de aceptación : *B, C, D y E* debido a que la regla establece que :

- *si un estado s del nuevo AFD contiene al estado final f del AFND, s será un estado de aceptación.*

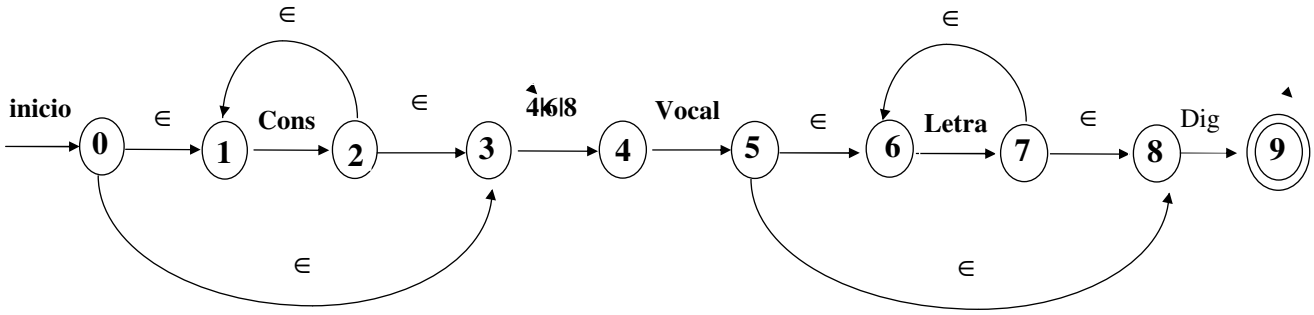
B = {1,2,3,4,6,9,12}
C = {2,3,4,5,6,8,9,11,12}
D = {2,3,4,6,7,8,9,11,12}
E = {2,3,4,6,9,10,11,12}



Los cuatro estados del AFD :
B,C,D y *E*, contienen al estado 12 !!.

- El estado final del AFND original es 12. Ver autómata 3.5.8

Ejemplo 3.10. *Encuentra el AFD para el siguiente AFND :*



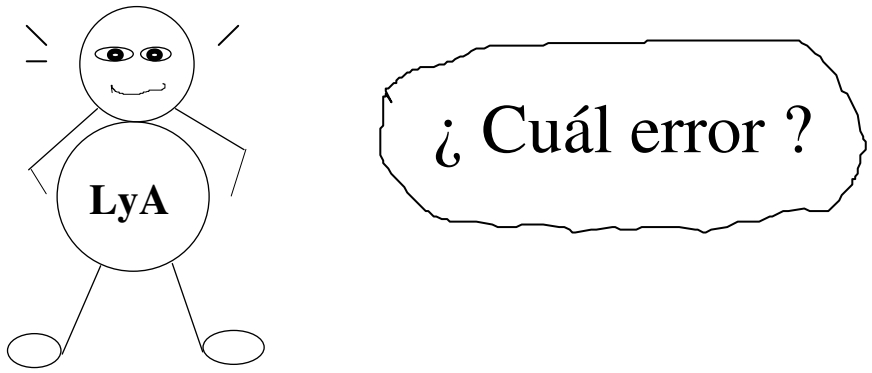
(3.10.1)

El AFND reconoce la expresión regular :

Cons ⇒ [B-DF-HJ-NP-TV-Zb-df-hj-np-tv-z]
Vocal ⇒ [A,a,E,e,I,i,O,o,U,u]
Letra ⇒ Cons | Vocal
Tres ⇒ 4 | 6 | 8
Dig ⇒ [0-9]
Token ⇒ (Cons) (Tres) (Vocal) (Letra*) Dig*

De inicio, debemos identificar los símbolos en la entrada. Un error común es seleccionar las siguientes columnas para la tabla D_{tran} :

Estados	Símbolos en la entrada				
	Cons	**Vocal**	**Letra**	**Tres**	**Dig**



La igualdad $\bigcup_{i=1}^{5} L(c_i) = \Sigma$ si cumple ya que :

L (Cons) **U** L (Vocal) **U** L (Letra) **U** L (Tres) **U** L(Dig) = Σ Alfabeto de entrada

y

$\Sigma = \{A, B,..., Z, a, b,..., z, 0, 1,..., 9\}$

Pero la intersección entre ellos, en algunos, no es el conjunto vacío. Es decir :

Cons $\cap$ *Letra* $\neq \emptyset$
Vocal $\cap$ *Letra* $\neq \emptyset$
Tres $\cap$ *Dig* $\neq \emptyset$

y no se cumple la regla : $c_i \cap c_j = \emptyset$; para $i \neq j$ .... *(r1)*

La correcta selección es la mostrada en la siguiente tabla D_{tran} :

Estados	Símbolos en la entrada			
	Cons	**Vocal**	**Tres**	**Otros**

Otros = {0,1,2,3,5,7,9}

Podemos comprobar fácilmente que $\bigcup_{i=1}^{4} L(c_i) = \Sigma$:

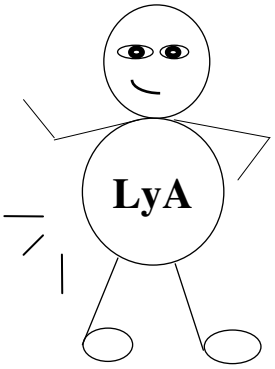
Σ = L (Cons) **U** L (Vocal) **U** L (Tres) **U** L(Otros)
Alfabeto de entrada

Y la regla *r1* se cumple. La dificultad ha sido salvada, debemos ahora iniciar con el algoritmo *de construcción de subgrupos* :

Recuerda que el primer paso es obtener :

$$A = \in\text{-}cerradura\ (s_0)$$

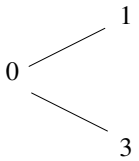
A es el estado inicial del nuevo AFD

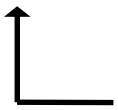


$$A = \in\text{-}cerradura\ (\{0\} = \{0,1,3\}$$

Símbolos en la entrada

Estados {0,1,3}		**Cons**	**Vocal**	**Tres**	**Otros**
	A				





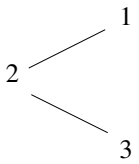
Añadimos *A* en la tabla D_{tran}

Luego, se calculan las transiciones del estado *A* para cada símbolo en la entrada.

Transiciones estado *A*.

- $\in\text{-}cerradura$ (*move*(A,Cons))
  move ({0,1,3},Cons) = {2} // Desde 1 alcanzamos el estado 2 con una *consonante*. Ver AFND 3.10.1.

$\in$*−cerradura* ({2}) = {2,3,1} // Nuevo estado *B*



- $\in$*−cerradura* (*move*(A,Vocal))
  move ({0,1,3},Vocal) = *Ø* // ver AFND 3.10.1

$\in$*−cerradura* (*move*(A,Vocal)) = "No existe" // No se genera nuevo estado.

- $\in$*−cerradura* (*move*(A,Tres))
  move ({0,1,3},Tres) = {4} // Del estado 3 alcanzamos al estado 4, con una lectura en la entrada de *4 | 6 | 8*. Ver AFND 3.10.1.

$\in$*−cerradura* ({4}) = {4} // Nuevo estado *C*

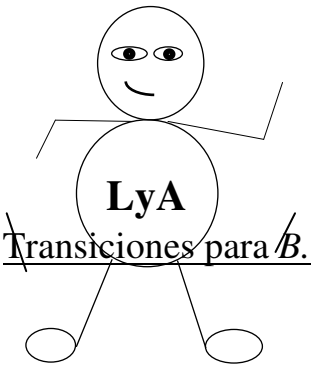
- $\in$*−cerradura* (*move*(A,Otro))
  move ({0,1,3},Otro) = *Ø* // ver AFND 3.10.1

$\in$*−cerradura* (*move*(A,Otro)) = "No existe" // No se genera nuevo estado.

Añadimos las transiciones del estado *A*, y los nuevos estados *B y C*, a la tabla D_{tran}.

Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B				
{4}	C				



Transiciones para *B*.

Ahora, debemos calcular las transiciones de los estados *B y C* para cada símbolo en la entrada. Aquí, podremos encontrar *nuevos estados* !!.

- $\in-$*cerradura* (*move*(B,Cons))
  move ({1,2,3},Cons) = {2} // Este move nos lleva al estado *B*

$\in-$*cerradura* ({2}) = <u>Estado *B*</u>

- $\in-$*cerradura* (*move*(B,Vocal))
  move ({1,2,3}, Vocal) = Ø // No hay transición

$\in-$*cerradura* (*move*(B,Vocal)) = "No existe" // Error

- $\in-$*cerradura* (*move*(B,Tres))
  move ({1,2,3}, Tres) = {4} // Estado *C*

$\in-$*cerradura* ({4}) = <u>Estado *C*</u>

- $\in-$*cerradura* (*move*(B,Otros))
  move ({1,2,3}, Otros) = Ø // No hay transición

$\in-$*cerradura* (*move*(B, Otros)) = "No existe" // Error

Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B	B	-	C	-
{4}	C				

D_{tran} con las transiciones para el estado B añadidos.

<u>Transiciones para *C*.</u>

- $\in-$*cerradura* (*move*(C,Cons))
  move ({1,2,3}, Cons) = Ø // No hay transición

$\in-$*cerradura* (*move*(C, Cons)) = "No existe" // Error

- $\in-$*cerradura* (*move*(C,Vocal))
  move ({4},Vocal) = {5} // El estado 4 alcanza al estado 5, con una entrada *vocal*. Ver AFND 3.10.1.

$\in-$*cerradura* ({5}) = {5,6,8} // <u>Nuevo estado *D*</u>

6

5

- $\in-$*cerradura* (*move*(C,Tres))
  move ({4}, Tres) = Ø // No hay transición

8

$\in-$*cerradura* (*move*(C, Tres)) = "No existe" // Error

- $\in-$*cerradura* (*move*(C,Otros))
  move ({4}, Otros) = $\emptyset$ // No hay transición

$\in-$*cerradura* (*move*(C, Otros)) = "No existe" // Error

Añadimos estas transiciones del estado *C* a la tabla D_{tran} :

Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B	B	-	C	-
{4}	C	-	D	-	-
{5,6,8}	D				

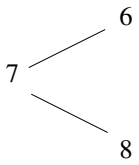
D_{tran} con las transiciones del estado *C* añadidas.

Ahora los estados *A, B y C* ya están "marcados" según el algoritmo de *construcción de subgrupos.* El estado *D* aún no está marcado, por lo tanto, tenemos que calcular sus transiciones.

Transiciones para *D*.

- $\in-$*cerradura* (*move*(D,Cons))
  move ({5,6,8}, Cons) = {7} // El estado 6 alcanza al estado 7, con una entrada *Cons*. Observar que *Cons* $\subset$ *Letra* Ver AFND 3.10.1.

$\in-$*cerradura* ({7}) = {6,7,8} // Nuevo estado E



- $\in-$*cerradura* (*move*(D,Vocal))
  move ({5,6,8}, Vocal) = {7} // El estado 6 alcanza al estado 7, con una entrada *Vocal*. Observar que *Vocal* $\subset$ *Letra*. Ver AFND 3.10.1.

$\in-$*cerradura* ({7}) = Estado *E*

- $\in-$*cerradura* (*move*(D,Tres))
  move ({5,6,8}, Tres) = {9} // El estado 8 alcanza al estado 9, con

una entrada *4|6|8*. Observa que *Tres* $\subset$ *Dig* .Ver AFND 3.10.1.

$\in$*−cerradura* ({9}) = {9} // Nuevo estado *F*

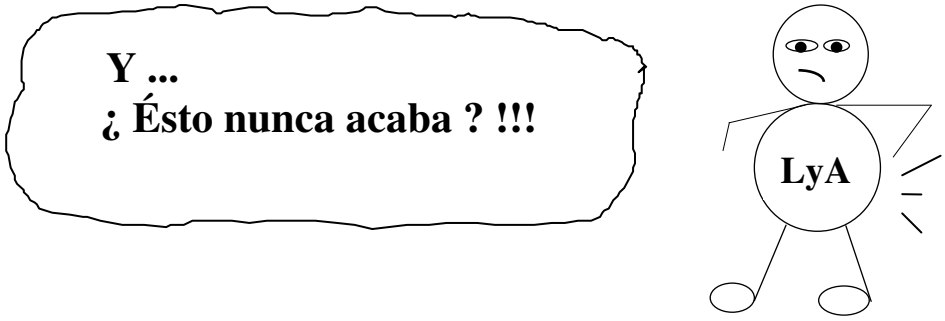
- $\in$*−cerradura* (*move*(D,Otros))
  move ({5,6,8}, Otros) = {9} // El estado 8 alcanza al estado 9, con una entrada *0|1|2|3|5|7|9* . Observa que *Otros* $\subset$ *Dig* . Ver AFND 3.10.1.

$\in$*−cerradura* ({9}) = Estado *F*

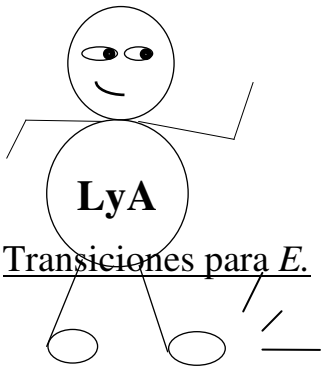
Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B	B	-	C	-
{4}	C	-	D	-	-
{5,6,8}	D	E	E	F	F
{6,7,8}	E				
{9}	F				

D_{tran} con la transicióm del estado *D* añadidos.



El algoritmo de *construcción de subgrupos* termina hasta que todos los estados en $D_{Estados}$ estén marcados, es decir, hasta que las transiciones para dichos estados estén calculadas.



Entonces, aún no hemos terminado, ya que falta de calcular las transiciones para los nuevos estados *E y F* .

Transiciones para *E*.

- $\in$*−cerradura* (*move*(E,Cons))
  move ({6,8,7}, Cons) = {7} // Este *move* nos lleva al estado *E*

$\in$*−cerradura* ({7}) = <u>Estado *E*</u>

- $\in$*−cerradura* (*move*(E,Vocal))
  move ({6,8,7}, Vocal) = {7} // Este *move* nos lleva al estado *E*

$\in$*−cerradura* ({7}) = <u>Estado *E*</u>

- $\in$*−cerradura* (*move*(E,Tres))
  move ({6,8,7}, Tres) = {9} // Este *move* nos lleva al estado *F*

$\in$*−cerradura* ({9}) = <u>Estado *F*</u>

- $\in$*−cerradura* (*move*(E,Otros))
  move ({6,8,7}, Otros) = {9} // Este *move* nos lleva al estado *F*

$\in$*−cerradura* ({9}) = <u>Estado *F*</u>

<u>Transiciones para *F*.</u>

- $\in$*−cerradura* (*move*(F,Cons))
  move ({9}, Cons) = $\emptyset$ // Transición no existe

- $\in$*−cerradura* (*move*(F,Vocal))
  move ({9}, Vocal) = $\emptyset$ // Transición no existe

- $\in$*−cerradura* (*move*(F,Tres))
  move({9}, Tres) = $\emptyset$ // Transición no existe

- $\in$*−cerradura* (*move*(F,Otros))
  move ({9}, Otros) = $\emptyset$ // Transición no existe

Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B	B	-	C	-
{4}	C	-	D	-	-
{5,6,8}	D	E	E	F	F
{6,7,8}	E	E	E	F	F
{9}	F	-	-	-	-

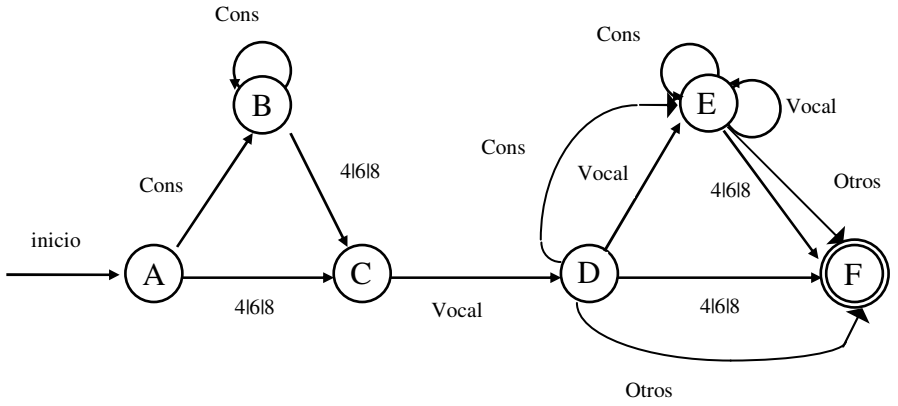
Ahora, todos los estados están marcados. En el anterior cálculo de transiciones para *E* y *F* no se generaron nuevos estados. Lo único que nos resta es construir el nuevo AFD a

D_{tran} con la transición del estado *E* y *F* añadidos.

partir de D_{tran} , pero antes simplificamos la tabla de transición, eliminando el estado *F* ya que no produce ninguna transición, es decir, no existen arcos que "salgan" de él.

Símbolos en la entrada

Estados		**Cons**	**Vocal**	**Tres**	**Otros**
{0,1,3}	A	B	-	C	-
{2,3,1}	B	B	-	C	-
{4}	C	-	D	-	-
{5,6,8}	D	E	E	F	F
{6,7,8}	E	E	E	F	F



(3.10.2)

Analicemos los estados del nuevo AFD :

A = {0,1,3}
B = {1,2,3}
C = {4}
D = {5,6,8}
E = {6,7,8}
F = {9} // *Estado de aceptación del nuevo AFD.*

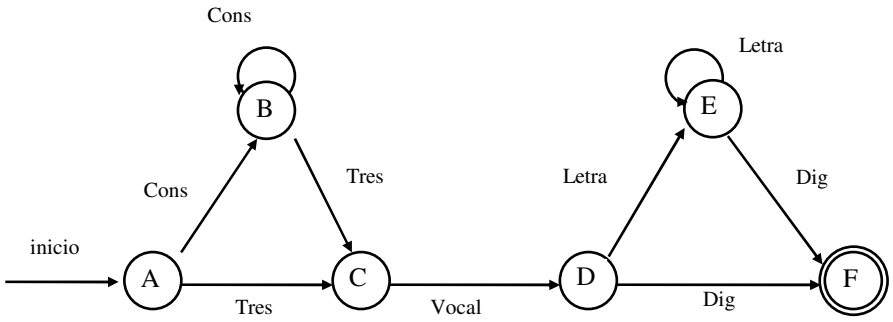
El único estado del nuevo AFD que contiene al estado *9* de aceptación del AFND es el **F**. Por lo tanto, el nuevo AFD sólo tiene un estado de aceptación : estado *F.*

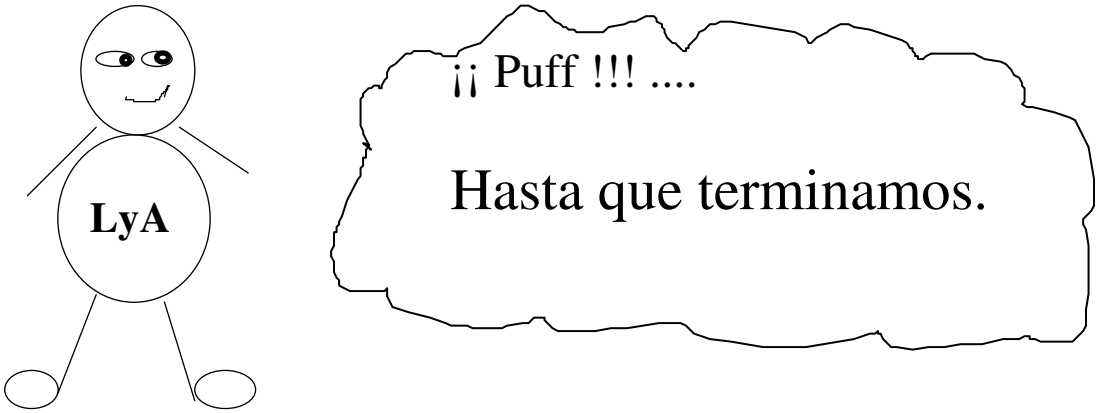
El autómata puede simplificarse en sus transiciones del estado *D* a *E*, del *E* al *E*, *E* a *F* y *D* a *F* ya que :

Cons | *Vocal* = *Letra*

4 | *6* | *8* | *Otros* = *Dig*

El AFD que reconoce la expresión regular es :





3.6 OPTIMIZACIÓN DE UN AUTÓMATA FINITO DETERMINÍSTICO AFD.

Cualesquier conjunto regular (aquél que es denotado por una expresión regular), es reconocido por un AFD, con un mínimo de estados. En esta sección mostraremos como construir un AFD reducido su conjunto *S* de estados, a un número óptimo, sin afectar el lenguaje que éste reconoce.

El algoritmo que optimiza el número de estados de un AFD se le denomina *algoritmo por particiones*, Fig. 3.15.

Entrada

AFD

Proceso

1. Construir una partición inicial Π del conjunto S de estados del AFD, con dos grupos : El de estados de aceptación F y el de los estados de no aceptación S-F.

2. Obtener una nueva partición Π_{nueva} a partir de Π aplicando el sig. procedimiento :

for (cada grupo G de Π) Do
Begin
Particionar G en subgrupos, tales que 2 estados s y t de G estén en el mismo subgrupo **si y solo si** para todos los símbolos a de entrada, los estados s y t tienen transiciones para a hacia estados en el mismo grupo de Π.
Reemplazar G en Π_{nueva} por el conjunto de todos los subgrupos formados
End

3. Si $\Pi_{nueva} = \Pi$, entonces $\Pi_{final} = \Pi$ y continuar con la etapa 4. De otra manera repetir la etapa (2) con $\Pi := \Pi_{nueva}$.

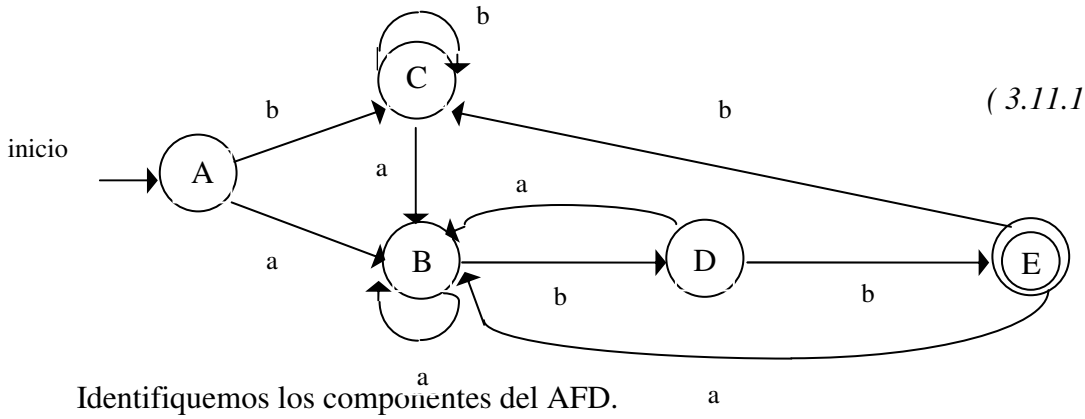
4. Seleccionar un estado de cada grupo de la partición final Π_{final} como el estado representativo de ese grupo.
Los estados representativos serán los estados del nuevo AFD mínimo u óptimo.

Salida

AFD mínimo

Fig. 3.15 Algoritmo de partición para optimizar un AFD.

Ejemplo 3.11 *Utilizando el algoritmo de partición, reduce u optimiza al siguiente AFD que reconoce la expresión regular (a | b)*abb.*



(3.11.1)

Identifiquemos los componentes del AFD.

Σ = {a,b} // Alfabeto de entrada

s_0 = A // Estado de inicio
F = {E} // Estados de aceptación
S = {A,B,C,D,E} // Estados del autómata

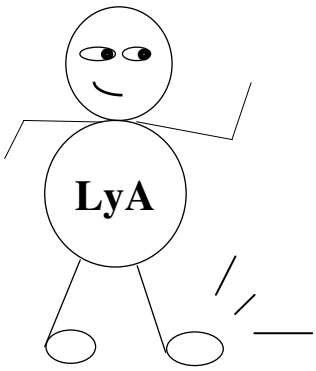
Función *move (s,a)*

Símbolos en la entrada

Estados

	a	**b**
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

(3.11.2)



Iniciaré la aplicación del algoritmo de partición.

El primer paso del algoritmo es la construcción de la partición inicial Π del conjunto **S** del AFD. Esta partición Π consta de dos grupos :

G_1 = {E} // Estados de aceptación; conjunto F del AFD
G_2 = {A,B,C,D} // Otros estados S-F

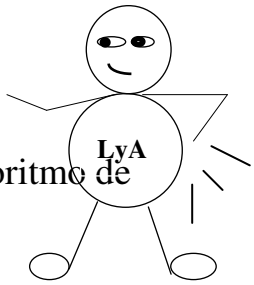
Podemos denotar la partición inicial Π de la siguiente manera :

Π = { (E) , (ABCD) }

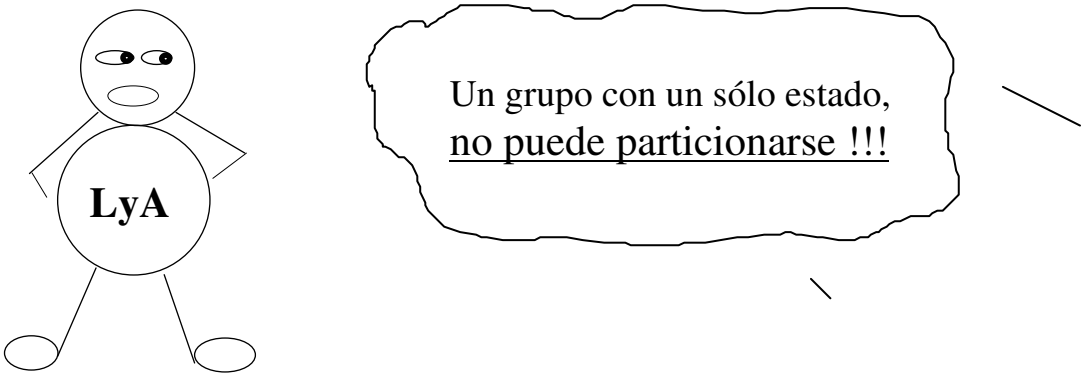
2o. Paso. Obtener una nueva partición Π_{nueva}. , pero ....

¡¡¿Cómo ?!!

- Pues aplicando el procedimiento usado en la etapa (2) del algoritmo de partición.



El grupo G_1 = (E) no es posible particionarlo y es obvio debido a que tiene un sólo estado.



Ahí, en Π_{nueva} se añade G_1. Π_{nueva} = { (E) }.

Veamos ahora el grupo G_2 = (ABCD).

- Transiciones con símbolo *a* de los estados en G_2. *:*

En todos los estados su transición es hacia el estado *B*, y este estado forma parte del grupo, por lo tanto, **NO HAY PARTICIÓN POSIBLE**.

$A \xrightarrow{a} B$

$B \xrightarrow{a} B$

$C \xrightarrow{a} B$

$D \xrightarrow{a} B$

- Transiciones con el símbolo b de los estados en G_2 :

$A \xrightarrow{b} C$

$B \xrightarrow{b} D$

$C \xrightarrow{b} C$

$D \xrightarrow{b} E$

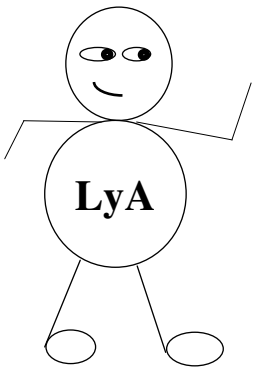
En este caso, los estados *A, B y C* transicionan hacia un estado perteneciente al grupo G_2, pero el estado *D* tiene una transición a un estado *E* que no forma parte de este grupo G_2. Por lo tanto **SI HAY PARTICIÓN** y ésta es :

G_{21} = (ABC) G_{22} = (D)

Agregamos estos nuevos grupos a Π_{nueva} :

$$\Pi_{nueva} = \{ (E) , (ABC) , (D) \}$$

La nueva partición ahora tiene 3 grupos : G_1 = (E), G_2 = (ABC) y G_3 = (D). Los grupos iniciales de Π = { (E), (ABCD) } han sido analizados, por lo tanto debemos seguir con la etapa (3) del algoritmo de partición.



Paso 3.

Comparamos Π_{nueva} con Π:

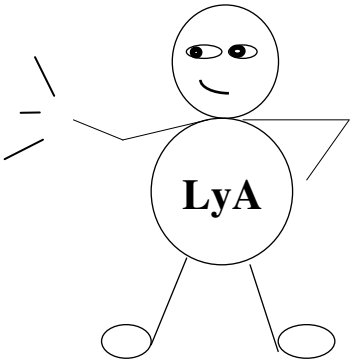
{ (E) , (ABC) , (D) } = { (E), (ABCD) } ?

¡ No son iguales !

Por lo tanto el paso (3) dice que debemos repetir la etapa (2), pero ahora con :

$$\Pi = \Pi_{nueva}$$

Bueno, aplicaremos de nuevo la etapa (2). Trataremos de encontrar mas particiones.



Con Π = { (E), (ABC), (D) }, el grupo que puede aceptar ser particionado es : (ABC). Analicemos las transiciones de los estados de este grupo para cada símbolo en la entrada:

Transiciones para a

$A \xrightarrow{a} B$

$B \xrightarrow{a} B$

$C \xrightarrow{a} B$

Transiciones para b

$A \xrightarrow{b} C$

$B \xrightarrow{b} D$

$C \xrightarrow{b} C$

En las transiciones para a , todos van a un estado de este grupo. Por lo tanto
no hay partición.

En las transiciones para el símbolo b , el estado *B* transiciona a un estado *D* que no pertenece a este grupo. Por lo tanto
si hay partición.

Entonces, la partición de (ABC) es : (AC) y (B). Estos nuevos subgrupos son añadidos a Π_{nueva} , en lugar del grupo (ABC) :

$$\Pi_{nueva} = \{ (E), (AC), (B), (D) \}$$

Enseguida probamos el paso (3) y vemos que :

$$\Pi \neq \Pi_{nueva}$$

Por lo tanto, repetimos el paso (2), con $\Pi := \Pi_{nueva}$.

Π = {(E), (AC), (B), (D)} es el valor para la partición que se somete al paso (2). El único grupo que puede particionarse es (AC). Analicemos pues, sus transiciones :

Transiciones para a

$A \xrightarrow{a} B$

$C \xrightarrow{a} B$

Transiciones para b

$A \xrightarrow{b} C$

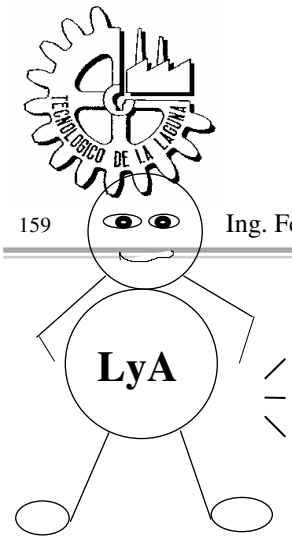
$C \xrightarrow{b} C$

Los estados *A* y *C* tienen transición hacia un estado *B* que no pertenece a su grupo.Si hacemos la partición, *A* y *C* quedarían en un mismo grupo ya que los dos transicionan al mismo estado *B*. Por lo tanto el grupo queda igual (AC).

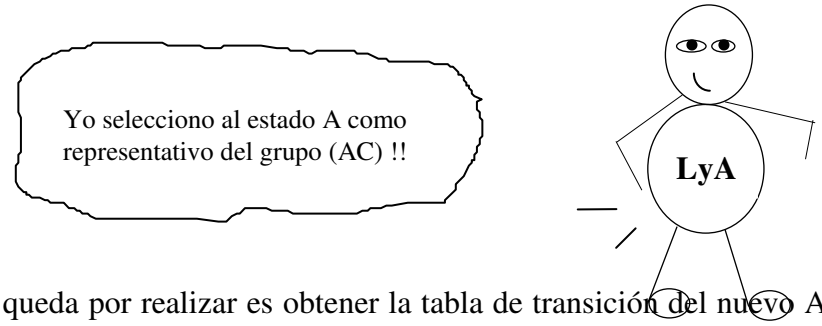
Los dos estados *A* y *C*, transicionan a un estado *C* que está dentro de su mismo grupo.
NO HAY PARTICIÓN

$$\Pi_{nueva} = \Pi ,$$

por lo tanto nos vamos al paso 4.



En el paso (4), de Π = {(E), (AC), (B), (D)} seleccionamos de cada grupo, <u>*un estado representativo*</u>. Es claro, que los estados representativos para los grupos (E), (B) y (D) son los mismos estados que componen a cada grupo. Pero en el grupo (AC) si debemos escoger a uno de los estados sea *A*, o bien *C*, como un estado representativo.



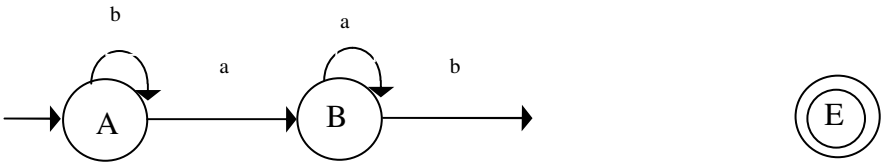
Lo que queda por realizar es obtener la tabla de transición del nuevo AFD mínimo. Los renglones de dicha tabla serán los estados representativos. Las transiciones son obtenidas del autómata 3.11.1.

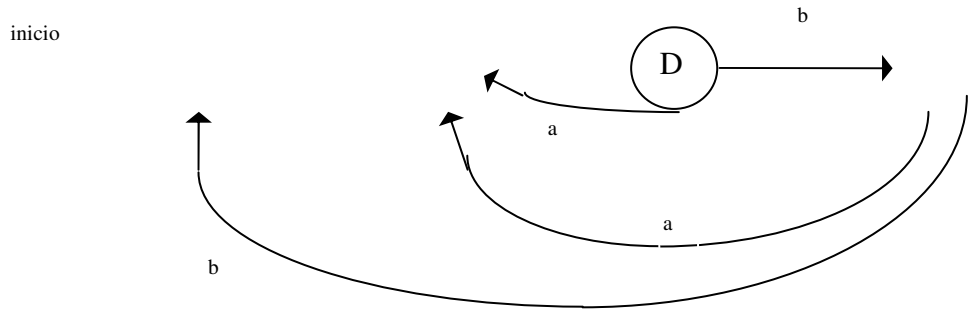
Símbolos en la entrada

Estados representativos		**a**	**b**
	A	B	A
	B	B	D
	D	B	E
	E	B	A

Observa que respecto a la tabla 3.11.2 , el estado *C* es sustituído por el estado *A*. El estado *C* ya no existe como renglón en la nueva tabla de transición del AFD reducido.

Y el AFD mínimo o reducido es el que se muestra a continuación :





Ejemplo 3.12. *Reducir a su óptimo el AFD 3.9.1 del ejemplo 3.9. El AFD 3.9.1 reconoce la expresión regular:*

$$Id \Rightarrow (Letra)\ (Letra|Dig|Sub)^*$$

Dándole un vistazo al AFD 3.9.1, vemos que el conjunto de estados S es :

S = {A, B, C, D, E}

donde A es el estado de inicio s_0 , y

F = {B, C, D, E} es el conjunto de estados de aceptación.

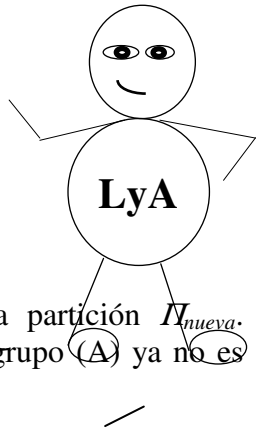
Aplicando el paso (1) del algoritmo de partición, tenemos la partición inicial Π es :

(A) // Estados de no aceptación S-F
(BCDE) // Estados de aceptación F

Entonces, Π = { (A) , (BCDE) }.

Continuamos con el paso (2).

Obtener una nueva partición : Π_{nueva}



El procedimiento del paso (2) nos dice como calcular la nueva partición Π_{nueva}. Observemos las transiciones de los estados del grupo (BCDE). El grupo (A) ya no es susceptible de partición.

- Transiciones símbolo *Letra*

$B \xrightarrow{\text{Letra}} C$

$C \xrightarrow{\text{Letra}} C$

$D \xrightarrow{\text{Letra}} C$

$E \xrightarrow{\text{Letra}} C$

Todos los estados transicionan a un estado *C,* y éste pertenece al mismo grupo. Por lo tanto :

No hay partición.

- Transiciones símbolo *Dig*

$B \xrightarrow{\text{Dig}} D$

$C \xrightarrow{\text{Dig}} D$

$D \xrightarrow{\text{Dig}} D$

$E \xrightarrow{\text{Dig}} D$

Todos los estados transicionan a un estado *D,* y éste pertenece al mismo grupo.

No hay partición.

- Transiciones símbolo *Sub*

$B \xrightarrow{\text{Sub}} E$

$C \xrightarrow{\text{Sub}} E$

$D \xrightarrow{\text{Sub}} E$

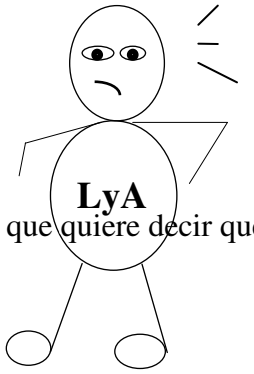
$E \xrightarrow{\text{Sub}} E$

Todos los estados transicionan a un estado *E,* y también como sucedió en los anteriores 2 casos, *E* pertenece al mismo grupo. Por lo tanto :

No hay partición.

Entonces,

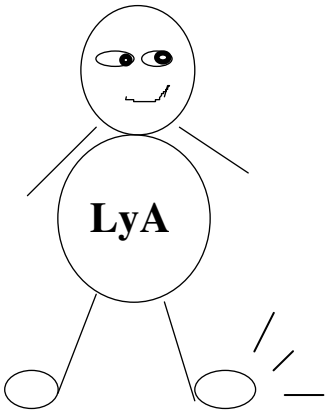
¿ No hay partición ?



Efectivamente en esta etapa no hubo partición, lo que quiere decir que :

$\Pi_{nueva} = \Pi = \{ (A) , (BCDE) \}$

Al aplicar el paso (3) la igualdad $\Pi_{nueva} = \Pi$ se cumple, por lo que la secuencia del algoritmo nos envía al paso (4), con $\Pi_{final} = \Pi$.



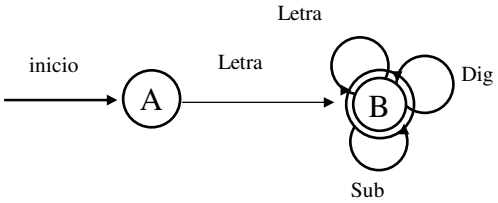
Paso 4. Seleccionar los estados representativos
de cada grupo de Π_{final} .

Selecciono del grupo (A) al propio A ,

Construimos la tabla de transición del nuevo AFD reducido, donde los renglones son los estados representativos.

Estados Representativos	Símbolos en la entrada		
	Letra	**Dig**	**Sub**
A	B	-	-
B	B	B	B

El diagrama del AFD mínimo, ahora se ha reducido de 5 estados a 2, y los arcos que antes eran 13 ahora sólo son 4, tal y como es mostrado enseguida.



AFD mínimo para :

Id $\Rightarrow$ *(Letra) (Letra|Dig|Sub)**

Ejemplo 3.13. *Construir el AFD reducido o mínimo, a partir del AFD 3.10.2 del ejemplo 3.10, (pag. 156).*

Haciendo referencia al AFD 3.10.2 encontramos que :

S = {A, B, C, D, E, F}

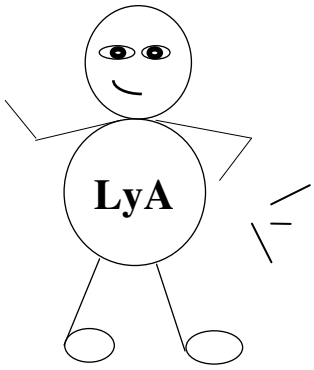
Del algoritmo de partición, el paso (1) es encontrar la partición inicial Π:

(ABCDE) // Estados de no aceptación, *S-F*
(F) // Estados de aceptación, *F*

De lo anterior, Π = { (ABCDE) , (F) }.

Ya tenemos la partición inicial, ahora debemos proceder a encontrar una nueva partición Π_{nueva}.

Paso 2.



El grupo (F) ya no tiene mas partición. Analicemos el grupo (ABCDE) para cada símbolo en la entrada. Existen 4 posibles entradas : *Cons*, *Vocal*, *Tres*, *Otros*; según el ejemplo 3.10. (ver tabla de transición de dicho ejemplo).

- Transición símbolo *Cons*

A ⟶ B

B ⟶ B

C // No tiene transición. Se agrega un "estado muerto" *Z*; C ⟶ Z

D ⟶ E

E ⟶ E

C transiciona a un estado *Z* que no se encuentra en el grupo (ABCDE). Por lo tanto <u>hay partición</u>. *(ABDE) y (C)*

- Transiciones símbolo *Vocal*

A ⟶ Z

B ⟶ Z

C ⟶ D

D ⟶ E

E ⟶ E

Ya que no existe transición, debemos mandar los estados *A* y *B* a un estado muerto *Z*.
La partición existe y ahora será formada por los grupos *(AB) (DE) (C)*

- Transiciones símbolo *Tres*

A ⟶ C

B ⟶ C

C ⟶ Z // Ya está hecha la partición

D ⟶ F // Los estados *D* y *E* tienen transición a un estado *F*, que no existe en el grupo. La partición ya está hecha.

E ⟶ F

- Transiciones símbolo *Otros*

A ⟶ Z

B ⟶ Z

C ⟶ Z

D ⟶ F

E ⟶ F

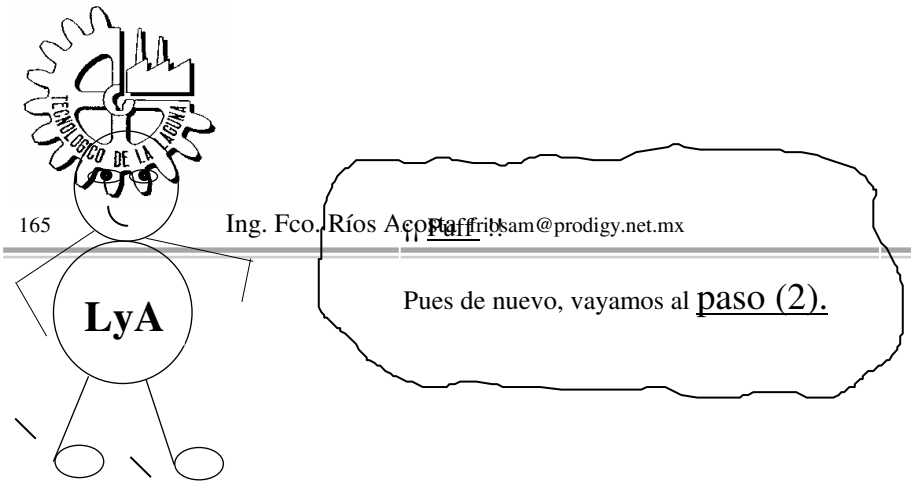
Los estados *A, B y C* transicionan a un estado muerto *Z y D,E* transicionan a un estado *F*. Tanto *Z* y *F* no pertenecen al grupo, por lo que la partición nueva es :

$$\Pi_{nueva} = \{ (AB) , (C) , (DE) , (F) \}$$

Al probar la condición en el paso (3) encontramos que :

$$\Pi_{nueva} \neq \Pi$$

Por lo que tenemos que repetir el paso (2), según el algoritmo de partición.



Ahora, $\Pi := \Pi_{nueva}$ = { (AB) (C) (DE) (F)}. Analicemos los grupos (AB) y (DE), ya que (F) y (C) no tienen partición posible.

- Grupo (AB)

TRANSICIONES			
Símbolo *Cons*	Símbolo *Vocal*	Símbolo *Tres*	Símbolo *Otros*
A ⟶ B	A ⟶ Z	A ⟶ C	A ⟶ Z
B ⟶ B	B ⟶ Z	B ⟶ C	B ⟶ Z

Observamos que *A* y *B* siempre transicionan a un mismo estado ya sea *B* o *Z*; por lo tanto no hay partición.

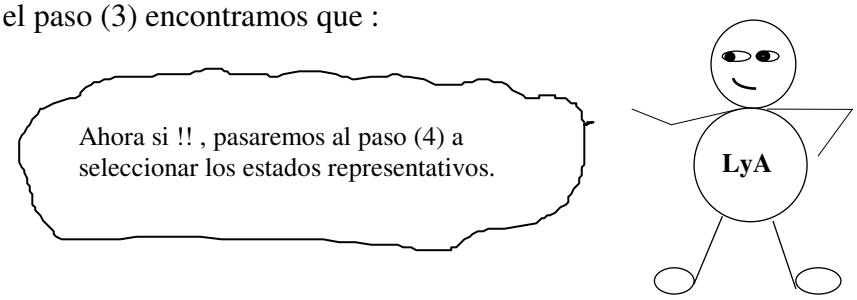
- Grupo (DE)

TRANSICIONES			
Símbolo *Cons*	Símbolo *Vocal*	Símbolo *Tres*	Símbolo *Otros*
D ⟶ E	D ⟶ E	D ⟶ F	D ⟶ F
E ⟶ E	E ⟶ E	E ⟶ F	E ⟶ F

Lo mismo observamos para los estados *D* y *E*. Siempre la transición es hacia un mismo estado, sea *E* o bien *F*. No hay partición

Asi que al probar la condición en el paso (3) encontramos que :

$$\Pi_{nueva} = \Pi$$



Paso (4).

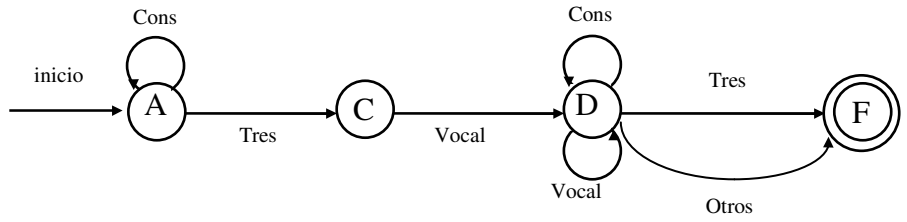
Grupo	Estado representativo
(AB)	A

(C)	C
(DE)	D
(F)	F

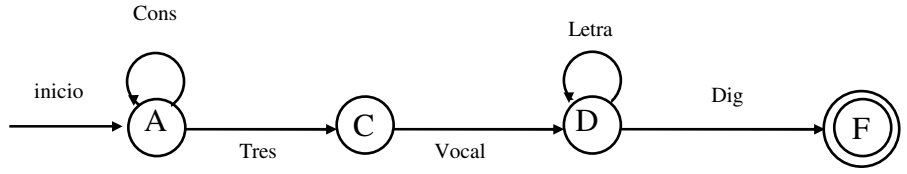
La tabla de transición del nuevo AFD mínimo es :

Estados Representativos	Símbolos en la entrada			
	Cons	**Vocal**	**Tres**	**Otros**
A	A	-	C	-
C	-	D	-	-
D	D	D	F	F

y el autómata AFD construido a partir de la anterior tabla, es el resultado buscado.



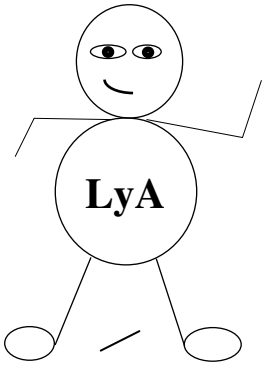
Sabemos que *Letra* $\Rightarrow$ *Cons* | *Vocal*; y que *Dig* $\Rightarrow$ *Tres* | *Otros*. Aplicamos estas definiciones al AFD y tenemos al nuevo diagrama del AFD mínimo o reducido :



3.7 PROPIEDADES DE LOS LENGUAJES ACEPTADOS POR UN AUTÓMATA FINITO.

En el capitulo 1 sección 1.5, se estableció que una expresión regular denota a un *lenguaje regular*. ¿ Qué quiere decir lo anterior ? Estamos ciertos, que una expresión regular se define sobre un alfabeto donde, el alfabeto Σ es un conjunto finito de símbolos. Entonces un *lenguaje regular* es un conjunto de cadenas formados a partir de los símbolos de Σ.

Un autómata finito es un reconocedor de cadenas, especificadas por una expresión regular. Por lo tanto, un *autómata finito* es un reconocedor de un *lenguaje regular*.



Un autómata finito reconoce solamente :

Lenguajes regulares .

Generalmente, un lenguaje regular es un subconjunto de la cerradura de un alfabeto, esto es :

$$L(r) \subset \Sigma^* \qquad (3.6.1)$$

Por definición una expresión regular denota un lenguaje regular (conjunto de cadenas), cuyas cadenas son una concatenación de símbolos de un alfabeto Σ. Y dado que la cerradura se define como :

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

donde :

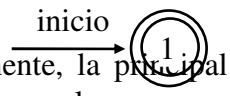
$\Sigma^0 = \{\in\}$

$\Sigma^1 = \{x \mid |x| = 1\}$ // Cadenas de longitud 1

$\Sigma^2 = \{x \mid |x| = 2\}$ // Cadenas de longitud 2

$\Sigma^3 = \{x \mid |x| = 3\}$ // Cadenas de longitud 3

Podemos ver que si $L(r) \subset \Sigma^*$, entonces un lenguaje regular, puede ser aquél que sólo tenga la cadena vacía $\in$. El autómata que reconoce a este lenguaje regular es :



El estado de inicio también es un estado de aceptación.

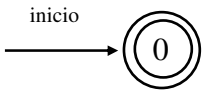
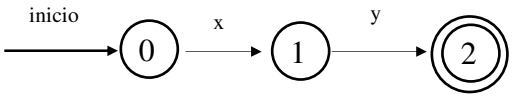
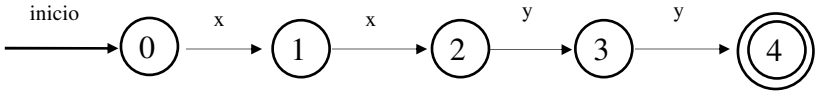
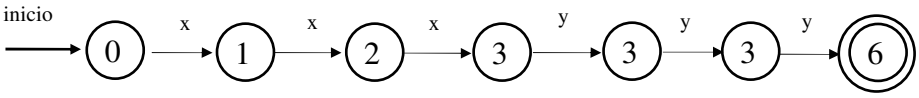
Finalmente, la principal propiedad ~~que debe cumplir un *lenguaje regular*, es~~ que las cadenas que lo componen puedan ser reconocidas por un autómata finito.

3.8 DETERMINACION DE LENGUAJES REGULARES Y NO REGULARES.

Algunos lenguajes *no pueden ser descritos por una expresión regular*. Las expresiones regulares no pueden ser usadas para describir construcciones anidadas o balanceadas. Por ejemplo, el conjunto de los paréntesis debidamente balanceados, no puede ser denotado por una expresión regular.

La repetición de cadenas es otro ejemplo de un lenguaje que no puede ser denotado por una expresión regular. Por ejemplo, sea $L = \{wcw \mid w$ es una cadena de a's y b's$\}$. Algunos lexemas de este lenguaje : aabcaab, aca, bcb, aaabbcaaabb, ... etc. El autómata finito debe reconocer una cadena *w* luego esperar por un símbolo *c*, para luego esperar la misma cadena que entró antes de la *c*. ¿Como guarda el autómata finito esa información para luego que *c* sea reconocido, esperar la misma cadena? **NO PUEDE HACERLO**. El autómata finito no puede, no tiene elementos para almacenar información de la entrada.

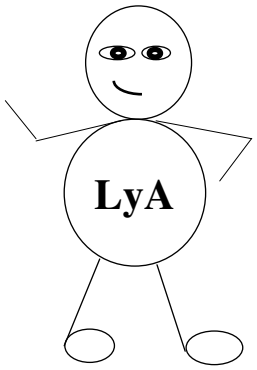
Otros ejemplo es el lenguaje $L = \{ x^n y^n \mid n \geq 0 \}$. Obtengamos los AFD's para diferentes valores de *n* :

n	AFD
0	
1	
2	
3	
...	

Conforme *n* sea un número entero lo suficientemente grande, el número de estados también se eleva en proporción *2*n,* tendríamos un autómata no programable debido al elevado número de estados que lo forman, cuando *n* es un número arbitrariamente grande.

Lenguajes como los anteriores, que no pueden ser reconocidos por un autómata finito, se denominan *lenguajes no regulares*.

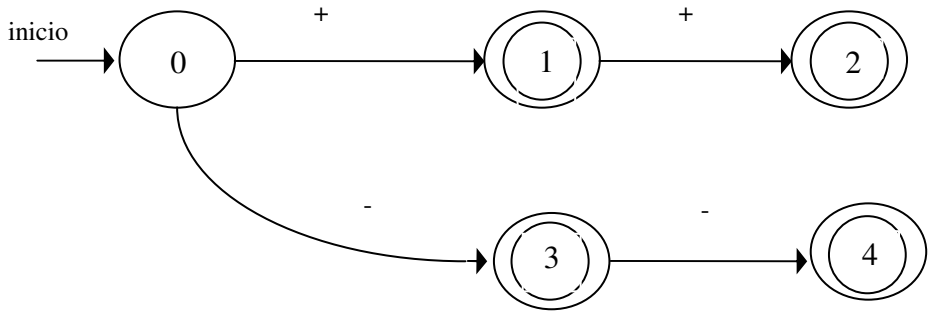
Si existe un autómata finito que reconozca a las cadenas de un lenguaje, éste es un *lenguaje regular* ; de lo contrario es un *lenguaje no regular.*



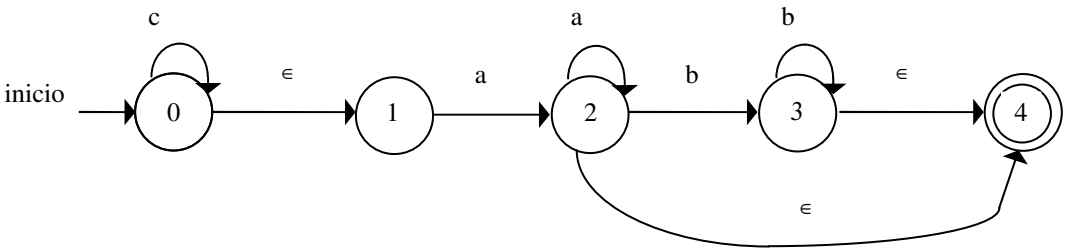
3.9 EJERCICIOS PROPUESTOS.

1. *Encontrar los componentes S, s_0 , F, Σ y la tabla de transición, para los siguientes autómatas :*

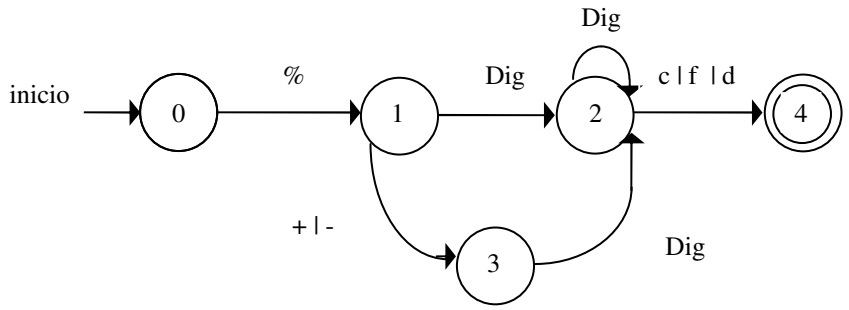
(a)



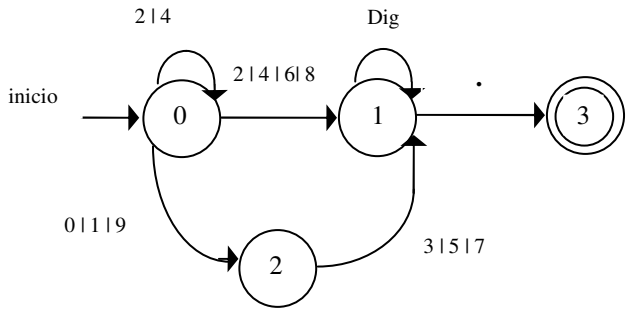
(b)



(c)



(d)

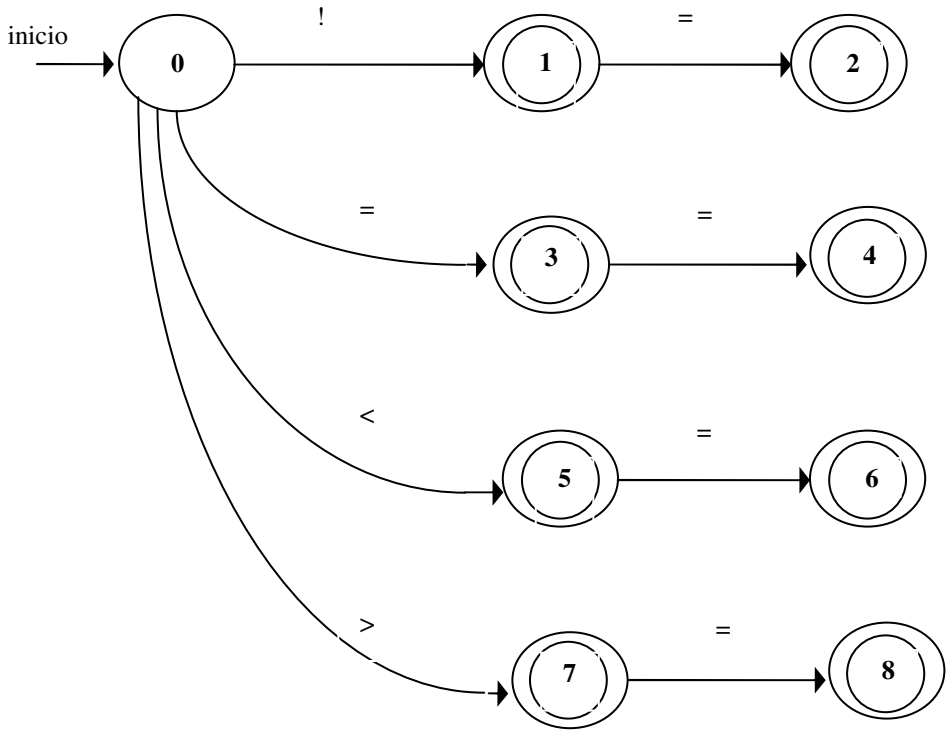


2. *¿ Cuáles de los autómatas en el ejercicio 1, son determinísticos y cuáles no determinísticos ? .*

3. *Aplica el algoritmo de la Fig. 3.5 para el siguiente AFD y cadenas de entrada :*

(a) !=

(b) <
(c) = =

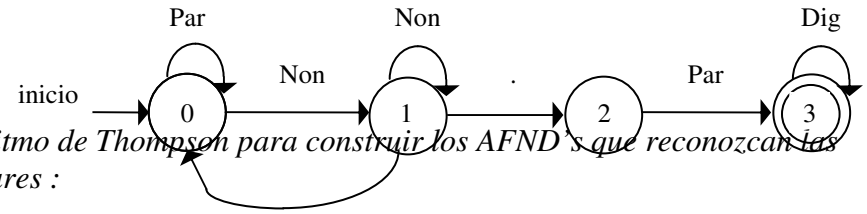


4. *Aplica el algoritmo de simulación para el AFD que se muestra y las siguientes cadenas de entrada :*

(a) 7651.27

(b) 3.8

(c) 4769.486



5. *Aplica el algoritmo de Thompson para construir los AFND's que reconozcan las expresiones regulares :*

(a) (Cons*) (4 | 6) (Vocal ?) (Letra*) (Dig)

(b) (w^+ x ?) (y ? | z*) ?

(c) (a | b $^{+}$) ? (c* d ?) *

(d) <= | < | > | >= | = = | !=

(e) (0 | 2 | 4) ? (a | b*) $^{+}$

(f) (Dig*) (Non) (Punto) (Par) (Dig*) // Punto $\rightarrow$ **.**

(g) (Letra*) (Dig*) (2) (1)

6. *Obtener los AFD's correspondientes a los ejercicios 5 a) hasta 5 g). Utiliza el algoritmo de construcción de subgrupos.*

7. *Minimiza los AFD's encontrados en el ejercicio 6, utilizando el algoritmo de minimización por particiones sucesivas.*