Industrial Data
ISSN: 1560-9146
iifi@unmsm.edu.pe
Universidad Nacional Mayor de San Marcos
Perú

Ruiz L., Edgar; Raffo L., Eduardo
Conversión de un AFN a un AFD
Industrial Data, vol. 6, núm. 1, agosto, 2003, pp. 61-70
Universidad Nacional Mayor de San Marcos
Lima, Perú

Vol. (6) 1: pp. 61-70

CONVERSIÓN DE UN AFN A UN AFD

(1) Edgar Ruiz L.
(2) Eduardo Raffo L.

RESUMEN

El artículo presenta la conversión de un autómata finito no determinista (AFN) a un autómata finito determinista (AFD), haciendo uso de la construcción por subconjuntos. El algoritmo de construcción por subconjuntos se basa en la clausura transitiva o cerradura ? , la implementación se realiza mediante un programa en lenguaje C++ , cuyo código y salida se presentan en su totalidad.

Palabras Claves: Autómata finito no determinista. Autómata finito determinista. Grafo de transiciones. Construcción de subconjuntos.

ABSTRACT

This article presents the change from a non-determinist fnite automaton (AFN) into a determinist finite automaton (AFD), making use of a subset construction. The subset construction algorithm is based on the transitive closure or ? ?lock. Its implementation is done through a C++ language program, whose code and output are thoroughly presented.

Key Words: Non-Determinist Finite Automaton. Determinist Finite Automaton. Transition graph. Subset construction.

AUTÓMATAS FINITOS(*)

Un reconocedor de un lenguaje es un programa que toma como entrada una cadena “x y”, responde "si" si x es una frase del programa, y "no", si no lo es. Se compila una expresión regular en un reconocedor construyendo un diagrama de transiciones generalizado llamado autómata finito. Un autómata finito puede ser determinista o no determinista, donde "no determinista" significa que en un estado se puede dar el caso de tener mas de una transición para el mismo símbolo de entrada.

Autómatas finitos no deterministas

Un autómata finito no determinista (abreviado, AFN) es un modelo formado por:

1. Un conjunto de estados denotados como: estados S.
2. Un conjunto de símbolos de entrada S (el alfabeto símbolos de entrada).
3. Una función de transición mover que transforma pares estado-símbolo en conjuntos de estados.
4. Un estado S0 que se considera el estado de inicio (o inicial).
5. Un conjunto de estados F considerados como estados de aceptación (o finales).

Un AFN se puede representar mediante un grafo dirigido etiquetado, llamado grafo de transiciones, en el que los nodos son los estados y las aristas etiquetadas representan las funciones de transición. Este grafo se parece a un diagrama de transiciones, pero el mismo carácter puede etiquetar dos o más transiciones fuera de un estado, y las aristas pueden etiquetarse con el símbolo especial ? ?y con símbolos de entrada.

En la Figura 1, se muestra el grafo de transiciones de un AFN que reconoce al lenguaje (a|b)*abb. El conjunto de estados del AFN es {0,1,2,3} y el alfabeto de símbolos de entrada es {a, b}. El estado 0 de la figura 1 se considera el estado de inicio, y el estado de aceptación 3 está indicado mediante un círculo doble.

Cuando se describe un AFN, se utiliza la representación de grafo de transiciones. En un computador, puede aplicarse la función de transición de un AFN de varias formas. La implantación más sencilla es una tabla de transiciones en donde hay una fila por cada estado y una columna por cada símbolo de entrada y ? , si es necesario.

---

(1) Docente del Departamento de Ingeniería de Sistemas e Informática.
Facultad de Ingeniería Industrial, UNMSM
E-mail: eruizl@unmsm.edu.pe
(2) Docente del Departamento de Ingeniería de Sistemas e Informática.
Facultad de Ingeniería Industrial, UNMSM
E-mail: eraffol@unmsm.edu.pe

(*) Los fundamentos teóricos han sido tomados de la primera referencia mencionada en la bibliografía. El programa en lenguaje C++ presenta la implementación de dichos algoritmos.

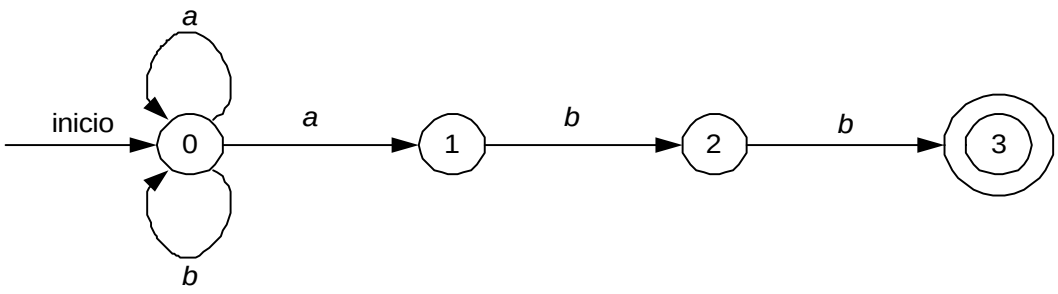

Figura 1. Un autómata finito no determinista

La entrada para la fila "i" y el símbolo "a" en la tabla es el conjunto de estados (o más probablemente en la práctica, un apuntador al conjunto de estados) que puede ser alcanzado por una transición del estado "i" con la entrada "a". En el Cuadro 1, se muestra la tabla de transiciones para el AFN de la Figura 1.

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

Ejemplo 1. En la Figura 2, se ve un AFN que reconoce aa*| bb*. La cadena *aaa* es aceptada recorriendo los estados 0,1,2,2 y 2. Las etiquetas de estas aristas son ? , *a*, *a* y *a*, cuya concatenación es aaa. Obsérvese, que los símbolos ? "desaparecen" en una concatenación.

Autómatas finitos deterministas

Un autómata finito determinista (abreviado, AFD) es un caso especial de un autómata finito no determinista en el cual:

1. Ningún estado tiene una transición ? , es decir, una transición con la entrada ? , y
2. Para cada estado s y cada símbolo de entrada a, hay exactamente una arista etiquetada a que sale de s.

Un autómata finito determinista tiene una transición desde cada estado con cualquier entrada. Si se está usando una tabla de transiciones para representar la función de transición de un AFD, entonces cada entrada en la tabla de transiciones es un solo estado. Como consecuencia es muy fácil determinar si un autómata finito determinista acepta o no una cadena de entrada, puesto que hay a lo sumo un camino desde el estado de inicio etiquetado con esa cadena.

Algoritmo 1: Simulación de un AFD

Entrada. Una cadena de entrada "x" que termina con un carácter de fin de archivo eof. Un AFD D con un estado de inicio "S0" y un conjunto "F" de estados de aceptación.

Salida. La respuesta "sí", si D acepta "x", "no", en caso contrario.

Método. Aplicar el algoritmo de la Figura 3 a la cadena de entrada "x". La función mover(s, c) da el estado al cual hay una transición desde el estado "s" en un carácter de entrada "c". La función "sgtecar" devuelve el siguiente carácter de la cadena de entrada "x".

Cuadro 1. Transiciones para el autómata finito de la figura 1

ESTADO	SIMBOLO DE ENTRADA	
	A	*B*
0	{ 0, 1}	{0}
1	-	{2}
2	-	{3}

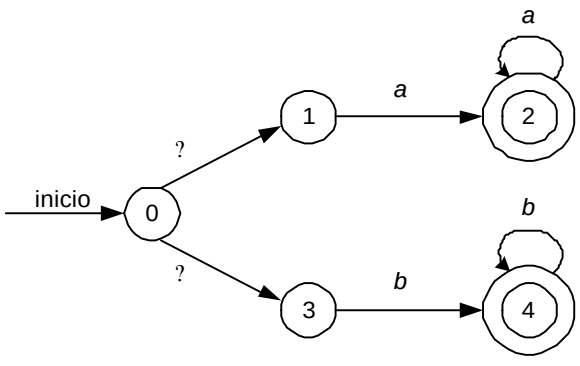

Figura 2. Un AFN que acepta *aa**| *bb**

```
s:=s0;
c:=sgtecar(car);
while c < > eof do
        s:=mover(s,c);
        c:=sgtecar(car);
end;
if s está en F then
        return "si"
else
        return "no";
```

Figura 3. Simulación de un AFD

Conversión de un AFN en un AFD

Se observa que el AFN de la Figura 1, tiene dos transiciones desde el estado "0" con la entrada "a"; es decir, puede ir al estado "0" o al 1. Igualmente, el AFN de la Figura 3 tiene dos transiciones en ? desde el estado "0".

Ahora se introduce un algoritmo para construir a partir de un AFN un AFD que reconozca el mismo lenguaje. Este algoritmo se le conoce como construcción de subconjuntos, es útil para simular un AFN por medio de un programa de computador.

Algoritmo 2: Construcción de subconjuntos. Construcción de un AFD a partir de un AFN.

Entrada. Un AFN *N*.

Salida. Un AFD *D* que acepta el mismo lenguaje.

Método. El algoritmo construye una tabla de transiciones *tranD* para *"D"*. Cada estado del AFD es un conjunto de estados del AFN y se construye *tranD* de modo que *"D"* simulará en paralelo todos los posibles movimientos que *"N"* puede realizar con una determinada cadena de entrada.

Cuadro 2. Operaciones sobre los estados de un AFN

OPERACION	DESCRIPCIÓN
cerradura-? (*s*)	Conjunto de estados del AFN alcanzables desde el estado *s* del AFN con transiciones ? solamente.
cerradura-? (*T*)	Conjunto de estados del AFN alcanzables desde algún estado *s* en *T* con transiciones ? solamente.
mover(*T*, *a*)	Conjunto de estados del AFN hacia los cuales hay una transición con el símbolo de entrada a desde algún estado *s* en *T* del AFN.

```
al inicio cerradura-? (s0) es el único estadosD y no esta marcado;
while haya un estado no marcado T en estadosD do begin
        marcar T;
        for cada símbolo de entrada a do
        begin
                U:= cerradura-? (mover(T, a));
                If U no esta en estadosD then
                        añadir U como estado no marcado a estadosD;
                tranD(T, a):= U
        end
end
```

Figura 4. La construcción de subconjuntos

Se utilizan las operaciones del Cuadro 2, para localizar los conjuntos de los estados del AFN ("s" representa un estado del AFN, y "T", un conjunto de estados del AFN).

Antes de detectar el primer símbolo de entrada, *"N"* se puede encontrar en cualquiera de los estados del conjunto cerradura-? (s_0), donde s_0 es el estado de inicio de K. Supóngase que exactamente los estados del conjunto "T" son alcanzables desde s_0 con una secuencia dada de símbolos de entrada, y sea a el siguiente símbolo de entrada. Al ver a, *"N"* puede trasladarse a cualquiera de los estados del conjunto mover(T, a). Cuando se permiten transiciones-? , *N* puede encontrarse en cualquiera de los estados de cerradura-? (T, a), después de ver la "a".

Se construyen estados *"D"*; el conjunto de estados de *"D"*; y *tranD*, la tabla de transiciones de *"D"*, de la siguiente forma. Cada estado de *"D"* corresponde a un conjunto de estados de AFN en los que podría estar *"N"* después de leer alguna secuencia de símbolos de entrada, incluidas todas las posibles transiciones- ? anteriores o posteriores a la lectura de símbolos.

El estado de inicio de *"D"* es cerradura-? (s_0). Se añaden los estados y las transiciones a *"D"* utilizando el algoritmo de la Figura 4. Un estado de *"D"* es un estado de aceptación si es un conjunto de estados de AFN que contenga al menos un estado de aceptación de *"N"*.

El cálculo de cerradura-? (T) es un proceso típico de búsqueda en un grafo de nodos alcanzables desde un conjunto dado de nodos. En este caso, los estados de "T" son el conjunto dado de nodos, y el grafo esta compuesto solamente por las aristas del AFN etiquetadas por ? .

Un algoritmo sencillo para calcular cerradura-? (T) utiliza una estructura de datos tipo pila para guardar estados en cuyas aristas no se hayan buscado transiciones etiquetadas con ? . Este procedimiento se muestra en la Figura 5.

```
meter todos los estados de T en pila;
inicializar cerradura-? (T) a T;
while pila no esté vacía do
begin
        sacar t, el elemento del tope, de pila;
        for cada estado u con una arista desde t a u etiquetada con ? do
                if u no esta en cerradura-? (T) do
                 begin
                        añadir u a cerradura-? (T);
                                meter u en pila
                 end
        end
end
```

Figura 5. Cálculo de cerradura-?

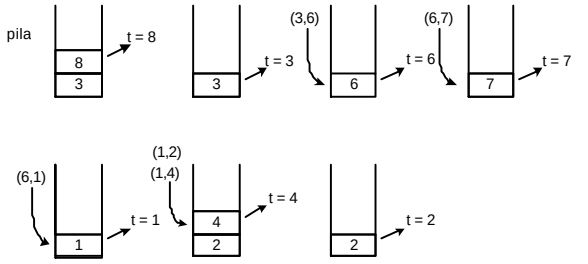

Figura 7. *tranD*[A, a] = B

Implementación de los algoritmos para convertir un AFN en un AFD

Para la implementación de estos algoritmos se utiliza el AFN *"N"* de la Figura 6 que acepta el lenguaje (a|b)*abb.

Se aplica el Algoritmo 2. Construcción de subconjuntos a *"N"*. El estado de inicio del AFD equivalente es cerradura-? (0), que es A = {0,1,2,4,7}, puesto que estos son alcanzados desde el estado 0 por un camino en que todas las aristas están etiquetadas por ? . El alfabeto de símbolos de entrada es {a, b}. Ahora el algoritmo indica que debe marcarse "A" y después calcular

cerradura-? (mover(A, a))

Calculando primero mover(A, a), el conjunto de estados de *"N"* que tiene transiciones en "a" desde miembros de "A". Entre los estados 0,1,2,4 y 7 sólo 2 y 7 tienen dichas transiciones, a 3 y a 8, de modo que:

cerradura-? (mover({0,1,2,4,7}, a)) =
cerradura-? (3,8) = {1,2,3,4,6,7,8}

Este conjunto se denominará "B". Así, tranD[A, a] = B. En la Figura 7, se presenta el cálculo de la cerradura-? . Entre los estados de "A", sólo 4 tienen una transición en "b" a 5, de modo que el AFD tiene una transición en "b" desde "A a".

cerradura-? (A, b)

cerradura-? (mover(A, b)) = *cerradura*-? ({5})

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

Por lo que, *tranD*[A, b] = C. Es decir; C = {1,2,4,5,6,7}

Se continúa este proceso con los conjuntos B y C, ahora sin marcar, finalmente se llegará al punto en que todos los conjuntos que son del estado AFD estén marcados.

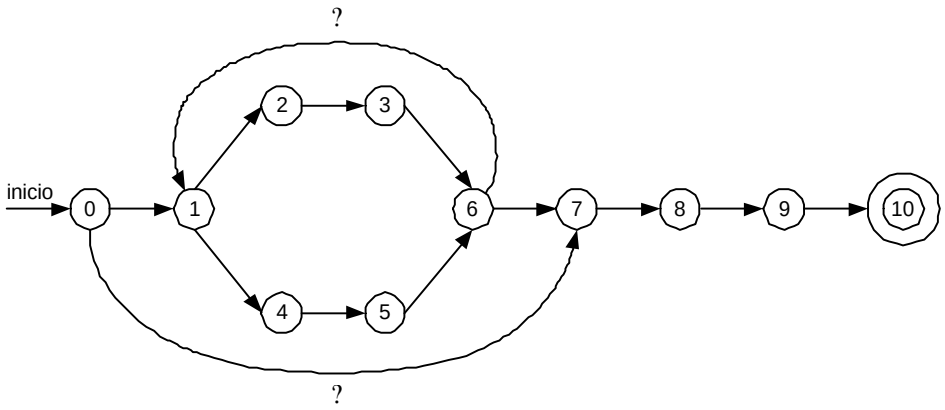

Figura 6. AFN *N* que acepta (a|b)*abb

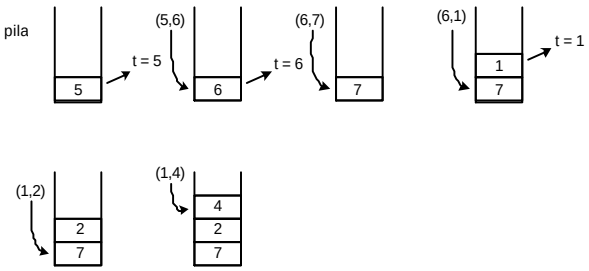

Figura 8. *tranD*[A, b] = C

Los dos estados siguientes son:

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

y

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

Resumiendo los cinco conjuntos de estados construidos son:

A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
E = {1,2,4,5,6,7,10}

Cuadro 3. Transiciones *tranD* para el AFD resultante

ESTADO	SIMBOLO DE ENTRADA	
	a	*b*
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

El estado "A" es el estado de inicio, y el estado "E" es el único estado de aceptación. La tabla de transiciones completa *tranD* se muestra en el Cuadro 3.

Finalmente en la Figura 9 se muestra el grafo de transiciones para el AFD resultante.

EL PROGRAMA

El programa AFNAFD.CPP se ha escrito en lenguaje C++, usando el compilador Turbo C/C++ 3.0 para MS-DOS. Se emplea la estructura de datos Pila. El símbolo ? (epsilon) se define a -1 como una constante simbólica. La matriz bidimensional de estados del AFN *"N"* se inicializa dentro del programa.

El código del programa se muestra en su totalidad en las subsiguientes figuras.

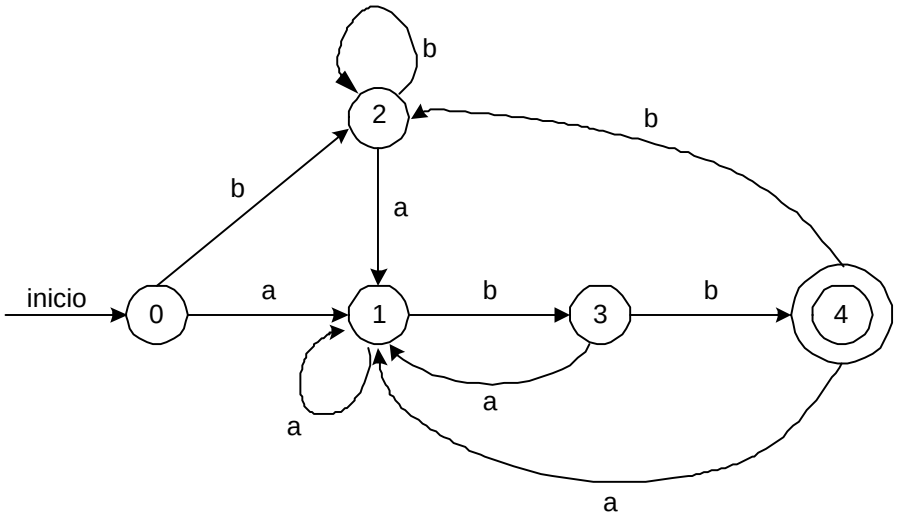

Figura 9. Resultado de aplicar la construcción de subconjuntos al AFN *N* de la figura 8

```
#include <iostream.h>
#include <iomanip.h>
#include <stdlib.h>
#include <string.h>
#include <conio.h>

#define AFN_MAX 11
#define EPSILON -1
#define ACEPTA  -2
#define STACKSIZE  100
#define MAXIMO 80

enum bool {FALSE,TRUE};
struct stack {
        int top;
        int items[STACKSIZE];
};
typedef stack *STACKPTR;

bool empty(STACKPTR ps);
int pop(STACKPTR ps);
void push(STACKPTR ps,int x);
void init(STACKPTR ps);
void e_clausura(int* ,int,int* ,int &na);
void mover(int *,int,int* t,int &m,int c);
void print(int* t,int m,char c);
void copy(int*,int,int estadosD[][AFN_MAX],int &nest,int*);
bool no_marcado(int*,int,int estadosD[AFN_MAX][AFN_MAX],int nest);
void aceptacion();
void print_estadosD(int estadosD[][AFN_MAX],int nest,int nD[]);

//          0  1  2  3  4  5  6  7  8  9 10
int aristas[][AFN_MAX]=
        {0, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,
         0,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,
         0,  0,  0,'a',  0,  0,  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,
         0,  0,  0,  0,  0,'b',  0,  0,  0,  0,  0,
         0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,
         0, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,'a',  0,  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,'b',  0,
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,'b',
         0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -2
        };

void main() // afnafd.cpp
{
    int m,i,j,k;
    int t[AFN_MAX];
    int estadosD[AFN_MAX][AFN_MAX],nest=0;
    int nD[AFN_MAX];
    int a[AFN_MAX],na;
    char* alfabeto="ab";
```

Figura 10. Código del programa: Primera parte

```
    clrscr();
    *t=0;m=1;
    cout << "estado : " << nest << endl;
    e_clausura(t,m,a,na);
    copy(a,na,estadosD,nest,nD);
    for(i=0;i<strlen(alfabeto);i++) {
        mover(a,na,t,m,alfabeto[i]);
  cout << "estado : " << nest << endl;
                print(t,m,alfabeto[i]);
                copy(t,m,estadosD,nest,nD);
    }
    for(k=1;k<nest;k++) {
            for(j=0;j<nD[k];j++)
                    t[j]=estadosD[k][j];
                    m=nD[k];cout << " K : " << k << endl;
          e_clausura(t,m,a,na);
                    for(i=0;i<strlen(alfabeto);i++) {
              mover(a,na,t,m,alfabeto[i]);
                            if(m) {
                                    if(no_marcado(t,m,estadosD,nest)) {
                                        cout << "estado : " << nest << endl;
                                        copy(t,m,estadosD,nest,nD);
                                        print(t,m,alfabeto[i]);
                                    }
                                    else
                                        print(t,m,alfabeto[i]);
                            }
                    }
            }
            aceptacion();
            print_estadosD(estadosD,nest,nD);
}
void print_estadosD(int estadosD[][AFN_MAX],int nest,int nD[])
{
            register int i,j;

            clrscr();
            cout << "  AFD                AFN              " << endl;
            cout << "---------------------------------------" << endl;
            for(i=0;i<nest;i++) {
                    cout << setw(4) << i << " : ";
                    for(j=0;j<nD[i];j++)
                            cout << setw(4) << estadosD[i][j];
                    cout << endl;
            }
            cout << "---------------------------------------" << endl;
            getch();
}
void aceptacion()
{
            cout << "estados de aceptacion tienen los nodos : ";
        for(int i=0;i<AFN_MAX;i++)
                    if(aristas[i][i]==ACEPTA)
                            cout << setw(4) << i;
            getch();
```

Figura 11. Código del programa: Segunda parte

```
}
void e_clausura(int* s,int n,int* a,int &na)
{
        int i,j,t,u;
        STACKPTR p;

        // meter todos los estados en pila
        // inicializar cerradura a
        init(p);
        na=0;
        for(i=0;i<n;i++) {
                push(p,s[i]);
                a[i]=s[i];
                na++;
        }
        while(!empty(p)) {
                t=pop(p);
         for(u=0;u<AFN_MAX;u++)
                        if(aristas[t][u]==EPSILON) {
                                i=0;
                                while(i<na && u!=a[i])
                                        i++;
                                if(i==na) {
                                    // a¤adir u a cerradura
                                    a[na++]=u;
                                     // meter u a pila
                                    push(p,u);
                                }
                        }
        }
        cout << " T         : " ;
        for(j=0;j<na;j++)
                cout << setw(4) << a[j];
        cout << endl;
}
void print(int* t,int m,char c)
{
        register int j;
    cout << " mover(T," << c << ") : ";
        for(j=0;j<m;j++)
                cout << setw(4) << t[j];
        cout << endl;
}
void copy(int* t,int m,int estadosD[][AFN_MAX],int &nest,int* nD)
{
        register int i;

        for(i=0;i<m;i++)
                estadosD[nest][i]=t[i];
        nD[nest]=m;
        ++nest;
}
void mover(int* a,int na,int* t,int &m,int c)
{
```

Figura 12. Código del programa: Tercera parte

```
	int i,j,k;

	m=0;
	for(i=0;i<na;i++) {
		k=a[i];
	 for(j=0;j<AFN_MAX;j++)
			if(aristas[k][j]==c)
				t[m++]=j;
	}
}
bool no_marcado(int* t,int m,int estadosD[][AFN_MAX],int nest)
{
	int k=0,j,i;

	for(k=0;k<nest;k++) {
		i=0;
		for(j=0;j<m;j++)
			if(t[j]==estadosD[k][j])
				i++;
		if(i==m)
			return FALSE;
	}
	return TRUE;
}
bool empty(STACKPTR ps)
{
	if(ps->top==-1)
		return TRUE;
	else
		return FALSE;
}
int pop(STACKPTR ps)
{
	if(empty(ps)) {
		cout << "\n pila vacia " << endl;
		exit(1);
	}
	return (ps->items[ps->top--]);
}

void push(STACKPTR ps,int x)
{
	if(ps->top==STACKSIZE-1) {
		cout << "\n stack overflow" << endl;
		exit(1);
	}
	else
		ps->items[++(ps->top)]=x;
}
void init(STACKPTR ps)
{
	ps->top=-1;
}
```

Figura 13. Código del programa: Cuarta parte

```
estado      : 0
 T                  : 0     1       7       2       4
estado      : 1
 mover (T, a)       : 8     3
estado      : 2
 mover (T, b)       : 5
 k   :  1
 T                  : 8     3       6       1       7       2       4
 mover (T, a)       : 8     3
estado      : 3
 mover (T, b)       : 9     5
 k   :  2
 T                  : 5     6       1       7       2       4
 mover (T, a)       : 8     3
 mover (T, b)       : 5
 k   :  3
 T                  : 9     5       6       1       7       2       4
 mover (T, a)       : 8     3
estado      : 4
 mover (T, b)       : 10    5
 k   :  4
 T                  : 10    5       6       1       7       2       4
 mover (T, a)       : 8     3
 mover (T, b)       : 5

estados de aceptación tienen los nodos: 10

            AFD                         AFN
-------------------------------------------------------------------
0           :       0       1       7       2       4
1           :       8       3
2           :       5
3           :       9       5
4           :       10      5
-------------------------------------------------------------------
```

Figura 14. Salida del programa AFNAFD.CPP

CONCLUSIONES

Este algoritmo conocido como el de construcción de subconjuntos convierte un AFN en un AFD haciendo uso de tres operaciones sobre los estados de un AFN y una pila. Lo importante del algoritmo es que se procesa el conjunto de estados del AFN denominado "T", usando operaciones de una estructura llamada pila.

El algoritmo sistemáticamente irá encontrando los nuevos estados de AFD usando únicamente los símbolos de entrada.

Si se desea correr el programa con un AFN diferente se debe inicializar la matriz de estados dentro del programa. Esto es conveniente cuando el AFN a convertir tiene un número de estados mayor de 5; pues de lo contrario se tendrían que ingresar estos por teclado lo cual aumenta la probabilidad que se introduzcan datos incorrectos. Otra alternativa cuando la entrada es muy grande, es guardar los datos de entrada en un archivo para luego ser tomados por el programa.

BIBLIOGRAFÍA

1. **Aho, Alfred V.; Sethi, Ravi; Ullman, Jeffrey D. (1990),** Compiladores Principios, técnicas y herramientas. Editorial Addison - Wesley Iberoamericana S.A., USA.

2. **Holu, Allen I. (1990),** "Compiler Design in C". Editorial Prentice Hall, USA.