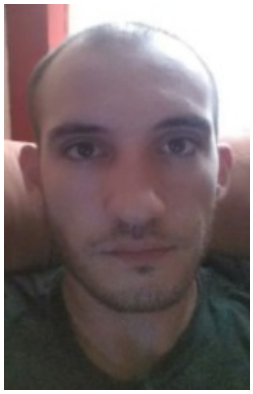


MODELOS DE COMPUTACIÓN

MEMORIA

DE

PRÁCTICAS

Corregidas

González Novez, Jesús Ángel
Grupo B2

Índice de contenido

Práctica 1....3
Práctica 2....5
Entrega de clase I....6
Práctica 3....8
Práctica 4....9
Práctica 5....10
Codificador:....10
Decodificar:....11
Práctica 6....12
Práctica 7....14

Práctica 1

Ejemplo 1 transparencia 55

G = (V , T , P, S), donde $V = \{S, A, B\}$, $T = \{a, b\}$ el símbolo de partida es S y las reglas de producción son:

$S \rightarrow aB$ $A \rightarrow bAA$ $S \rightarrow bA$ $B \rightarrow b$

$A \rightarrow a$ $B \rightarrow bS$ $A \rightarrow aS$ $B \rightarrow aBB$

Esta gramática genera el lenguaje :

$L(G) = \{u \mid u \in \{a, b\}^+$ y $N_a(u) = N_b(u)\}$

Donde $N_a(u)$ y $N_b(u)$ son el número de apariciones de símbolos a y b, en u, respectivamente.

Explicación de clase:

Lo que nos pide ese lenguaje o problema es generar una cadena de producción a partir de un punto de partida que en nuestro caso es S , y esa cadena producida debe cumplir que el número de “aes” sea igual al número de “bes” , donde a y b son símbolos terminales de la gramática G que representamos en minúscula.

El conjunto de símbolos terminales será para fabricar por ejemplo una tortilla de patatas o un coche (ingredientes), en la cadena producida no puede aparecer una variable de V, que sea conjunto de símbolos no terminales por ejemplo: sartén, olla, brazo robot para coche:

Si nos preguntamos ¿por qué no puede aparecer una variable en la cadena que fabricamos ?

Cuando nos vamos a Mercadona a comprar la tortilla de papas nunca la compramos con la sartén o la olla, al igual que cuando compramos el coche no lo compramos con el brazo de soldadura que cuesta un dineral.

Análisis de las reglas de producción:

El objetivo es generar una cadena de producción con el número de a igual que b si nos fijamos en la regla de producción $A \rightarrow aS$, tiene mas a que b, demostración:

$A \rightarrow aS \rightarrow abA \rightarrow aba$

Lo mismo para la palabra B que tiene mas b que a, pero la palabra S tiene el mismo número de a que b. Cuando generamos el mismo número de a que b ya tenemos la producción generada.

Generación de la cadena:

Empezamos desde S o cojo aB o bA

Si cojo aB tendríamos:

$S \rightarrow aB \rightarrow$ (como tenemos una regla que es $B \rightarrow b$) pues cambiamos B por b y obtenemos:

$S \rightarrow aB \rightarrow ab$

Ya tenemos una cadena generada con número de a igual que b sin aparecer ninguna variable.

Otra alternativa:

S → aB → (como quiero generar b pues cambiaría B por bS) → abS → abaB → abab

Ejemplo 2 Trasparencia 58

Sea G = ({S, X , Y }, {a, b, c}, P, S) donde P tiene las reglas,

S → abc

bY → Yb

S → aXbc

aY → aaX

Xb → bX

aY → aa

Xc → Ybcc

Esta gramática genera el lenguaje: $\{a^n b^n c^n \mid n = 1, 2, \ldots\}$.

Ese lenguaje nos dice que la cadena que vamos a generar a partir de ese lenguaje debe tener un número de a iguales que b y que c.

S → abc con esa gramática ya tenemos una cadena generada , porque el número de a es 1 como en el caso de b y de c.

Ahora partimos de otro estado inicial que es:

S → aXbc → (sustituimos Xb por bx) → abXc → (sustituimos Xc por Ybcc) → abYbcc(sustituimos bY por Yb) → aYbbcc(sustituimos aY por aa) → aabbcc

Hemos generado otra cadena que cumple con el lenguaje : el número de a es igual que de b y de c.

Práctica 2

Se pide determinar si la gramática G= ({S,A,B},{a,b,c,d},P,S) genera un lenguaje de tipo 3. Donde P es el conjunto de reglas de producción siguiente:

S → AB A → Ab A → a //Genera ab^i

B → cB B → d //Genera c^jd

Esta gramática genera por tanto el lenguaje L = {ab^ic^jd | i,j N}, es al menos libre de contexto.

Tenemos una gramática regular. Además este lenguaje se puede generar mediante la siguiente gramática:

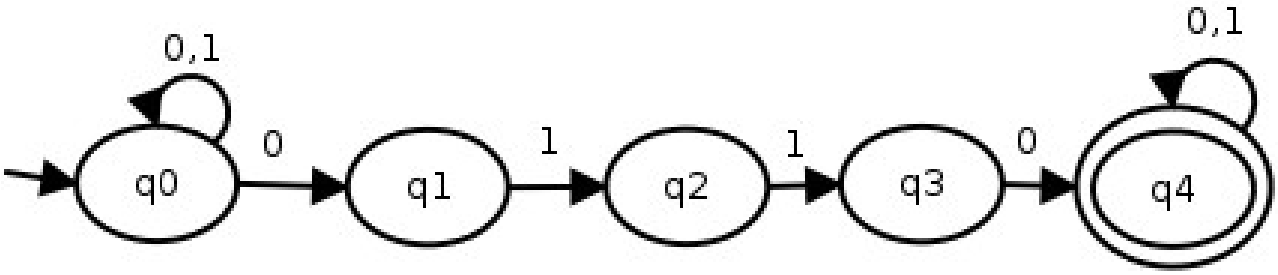
S → aB B → bB B → C c → cC C → d

Esta es de tipo 3, es decir es similar a la anterior, pero optimizada. Al ser de tipo 3 podemos afirmar que el lenguaje L también lo es.

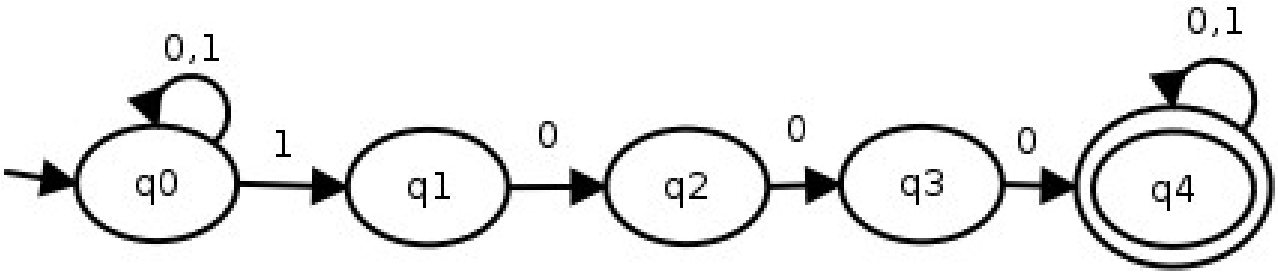
Entrega de clase I

Debemos realizar un autómata que nos sirva para las palabras que contenga las cadenas “0110” o bien “1000”

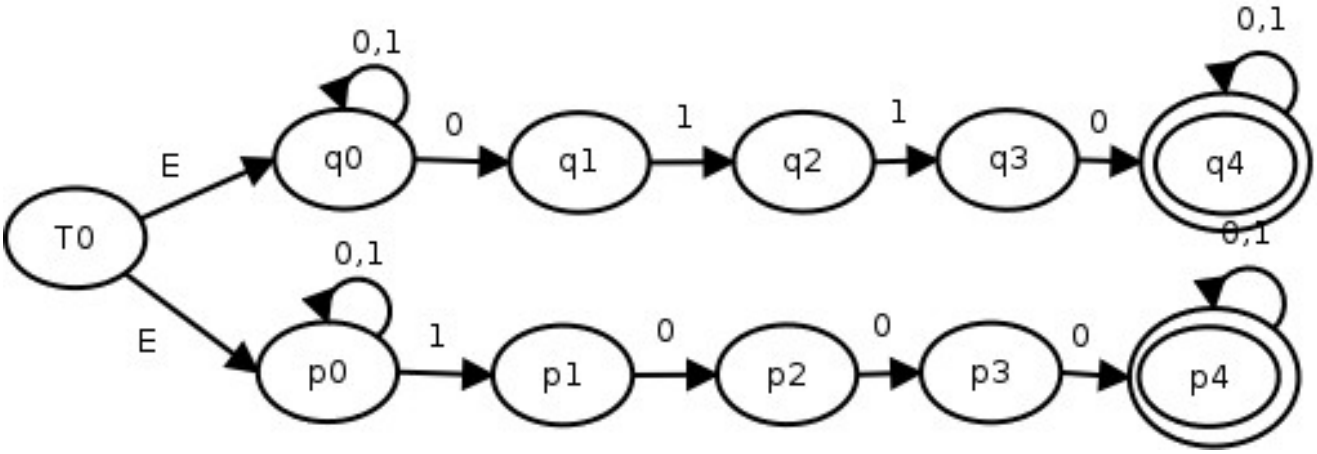
Primero vamos a ver para la cadena “0110”:



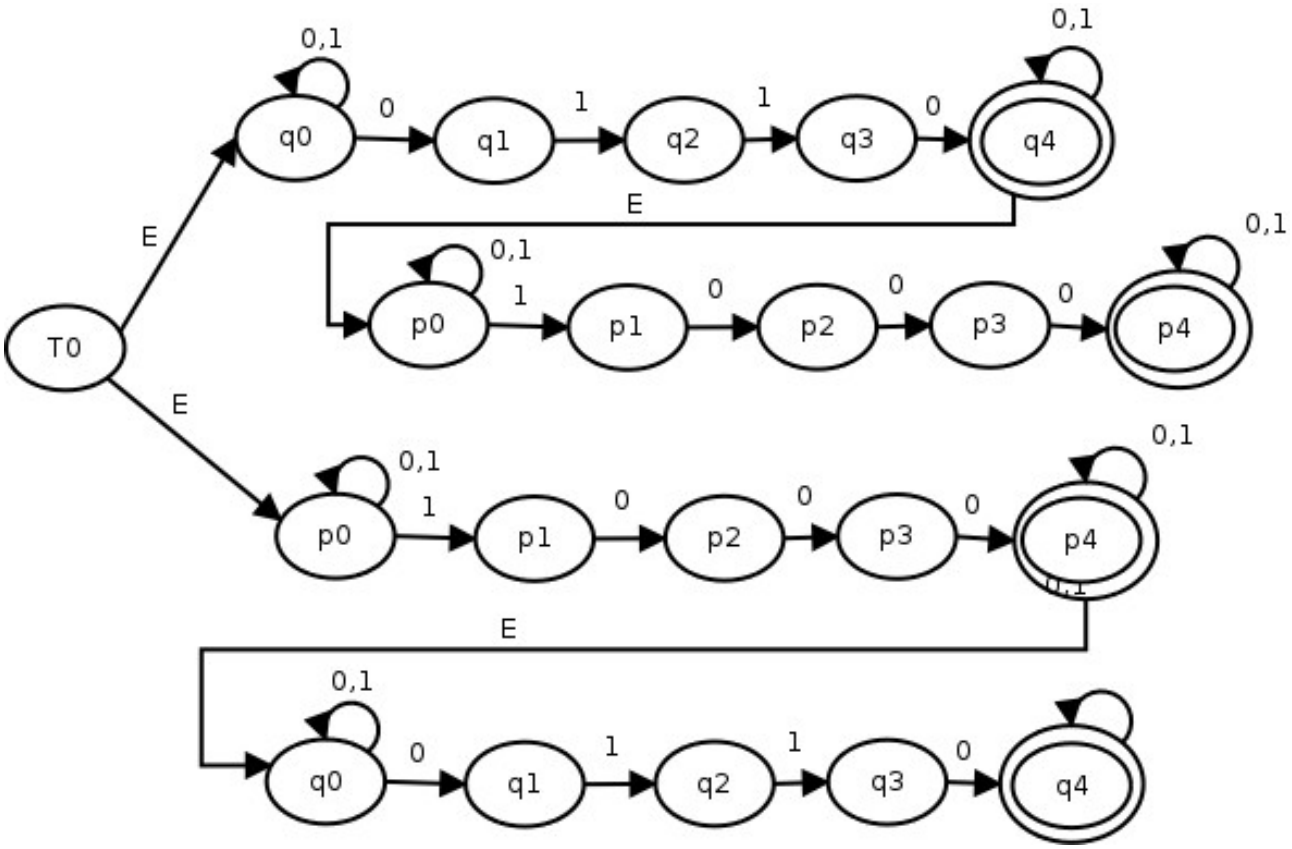
Y Ahora veamos para la cadena “1000”:



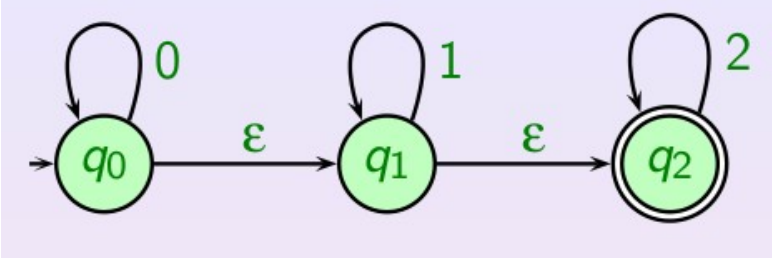
El objetivo es hacer un autómata válido para ambas cadenas, por tanto deberemos hacer uso de transiciones nulas:



Finalmente vamos a hacer que primero lea una cadena y después la otra sin importar el orden:



Práctica 3

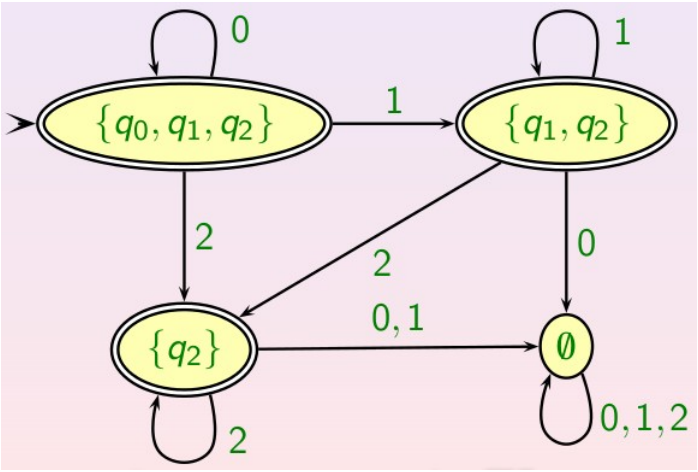


Se pide explicar el proceso que nos lleva de un autómata no determinista a uno determinista. Como vimos tenemos dos opciones:

- Eliminar las transiciones nulas (no recomendable)

- Plantear el autómata de forma que cualquier estado puede estar en el estado inicial. Esta opción es la que desarrollaremos.

Usaremos el concepto de unión para ver que acciones pueden hacerse en cada estado. Veamos como quedaría:



Este autómata hace exactamente lo mismo que el original, pero ya si es determinista. Debemos tener en cuenta que cada estado es la unión de los posibles estados, es decir, en el estado inicial analizamos los caminos y vemos que con el 0 volvemos al mismo estado que incluye q0 como hace el original, si leemos un 1 ya vamos a la parte q1,q2 descartando ya a q0, en este nuevo estado podemos leer un 1 y volver al mismo pues esta q1 o leer un 2 y avanzar al estado q2 en solitario.

También el primer estado podemos pasar a q2 directamente leyendo un 2. En q2 si leemos un 2 volvemos a q2. Como vemos el comportamiento es similar al del autómata original. Por otro lado vemos que esto no es “jauja”, es decir no podemos “volver hacia atrás” por tanto se plantean los casos de lectura de números correspondientes a cada estado que sean anterior a él mismo. Es decir en el estado {q1,q2} no es correcto leer un 0, ya pasó su momento digamos, por tanto iría al sumidero. Igual ocurre con el estado {q2} si lee un 0 ó un 1, va al sumidero también. Y por supuesto el sumidero si lee un 0,1 ó 2 vuelve a sí mismo. Queda definido así completamente nuestro autómata.

Práctica 4

Crear el fichero “ejemplo” con este contenido:

```
car     [a-zA-Z]
digito  [0-9]
signo   (\-|\+)
suc     ({digito}+)
enter   ({signo}?{suc})
real1   ({enter}\.{digito}*)
real2   ({signo}?\.{suc})
        int  ent=0, real=0, ident=0, sumaent=0;
%%
int i;
{enter}{
        ent++;
        sscanf(yytext," %d",&i);
        sumaent += i;
        printf("Numero entero %s\n",yytext);
        }
({real1}|{real2}){
        real++;
        printf("Num. real %s\n",yytext);
}
{car}({car}|{digito})*{
        ident++;
        printf(ïdentificador %s\n",yytext);
}
.|\n    {;}
%%
yywrap(){
        printf("Numero de Enteros %d, reales %d, ident %d,Suma de Enteros
%d",ent,real,ident,sumaent);
        return 1;
}
```

Ejecutar lex con el fichero creado:

lex ejemplo

Compilar el programa que crea lex:

gcc lex.yy.c -o prog -ll

Ejecutar el programa.

Práctica 5

El objetivo de esa practica codificar un mensaje de entrada usando una máquina de estados (máquina Mealy)

Una máquina de estados se basa en autómatas finitos y no tiene estado final, los autómatas finitos no producen salida.

En la teoría de la computación, una máquina de Mealy es un tipo de máquina de estados finitos que genera una salida basándose en su estado actual y una entrada. Esto significa que el diagrama de estados incluirá ambas señales de entrada y salida para cada línea de transición.

Codificador:

Primer símbolo 0 → 0

1 → 1

Resto

si antes hemos leído 0

si leemos un 0 → 0

si leemos un 1 → 1

si antes hemos leído 1

si leemos un 0 → 1

si leemos un 1 → 0

Podemos juntar lo anterior en dos estados Q0 y Q1

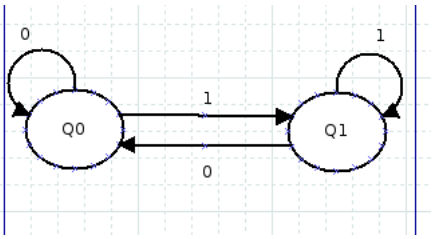
Si estamos en Q0 es porque se ha leído un 0

Si estamos en Q1 es porque se ha leído un 1

Aplicando lo anterior a un mensaje de entrada obtenemos como salida lo siguiente

1→1 0→1 0→0 0→0 1→1 0→1 1→1

El autómata finito que codifica esa entrada será:



1/1 0/0 0/1 1/0

Si aplicamos eso a la entrada 1000101:

Estamos en estado inicial Q0,leemos 1 salida es 1 y pasamos a Q1, ahora en Q1 leemos 0 salida 1 porque la máquina Mealy se basa en el estado actual y la entrada, pasamos a Q0, estando en Q0

leemos 0 y como antes hemos leído un 0 la salida es 0 y seguimos en Q0, de nuevo leemos un 0 y como se ha leído un 0 la salida es 0 y seguimos en Q0, leemos un 1 pasamos a Q1 y la salida es 1, leemos un 0 pasamos a Q0 y la salida es 1, leemos un 1 pasamos a Q1 y la salida es 1.

La salida de 1000101 es 1100111

Decodificar:

Decodificar con maquina de estados la maquina de estado no tiene estado final:

Primer símbolo

0 → 0

1 → 1

Resto

Si Ante-escrito 0

si leemos un 0 → 0

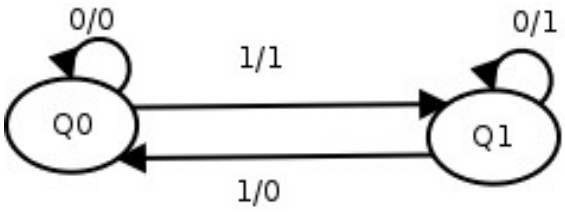
si leemos un 1 → 1

si Ante-escrito 1

si leemos un 0 → 1

si leemos un 1 → 0

Cogemos la salida anterior 1100111 y la decodificamos usando un autómata finito



1/1 0/0 1/0 0/1

Q0= Ante-escrito es 0 (o primer símbolo es 0)

Q1= Ante-escrito es 1

Cuando leemos 1 la salida es 1 y pasamos a Q1, leemos el siguiente que es 1 y la salida es 0 porque ante-escrito es 1 y pasamos a Q0, leemos un 0 y la salida es 0 porque ante-escrito es 0 y nos quedamos en Q0, leemos el siguiente 0 y da salida 0 porque ante-escrito es 0 y nos quedamos en Q0, leemos el 1 y la salida es 1 porque ante-escrito es 0 y pasamos a Q1, leemos el siguiente 1 y la salida es 0 porque ante-escrito es 1 y pasamos a Q0, leemos el último 1 y la salida es 1 porque el ante-escrito es 0 y pasamos a Q1, la salida de decodificador sera 1000101.

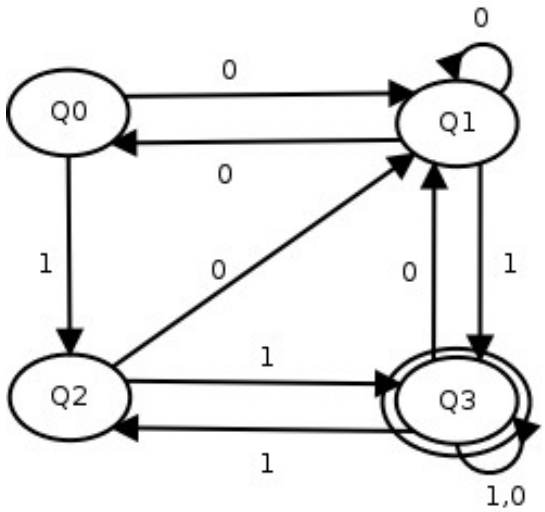
Práctica 6

El objetivo de esta practica

- El lenguaje aceptado por un autómata si es vacío o no
- El lenguaje aceptado por un autómata es finito o infinito

Para mostrar si el lenguaje aceptado por un autómata es vacío aplicamos el algoritmo eliminamos estados inaccesibles mediante un recorrido por el grafo a partir del estado inicial y comprobar si quedan estados finales.

Con este ejemplo de autómata:



Empezando en Q0 es posible ir a Q1 y desde ahí llegar a Q3, también posible desde Q0 llegar a Q2 y desde ahí a Q3, por tanto podemos decir que puede llegar al final y que no es un lenguaje vacío.

El segundo objetivo es :

El lenguaje aceptado por un autómata es finito o infinito para mostrar eso ,aplicamos el algoritmo:

Se suponen eliminados los estados inaccesibles y se eliminan los estados de error o estados desde los que no se pueden llegar a estados finales.

Eso lo hacemos recorriendo el grafo en sentido contrario a los arcos y empezando en los estados finales.

Comprobamos si en el grafo generado quedan ciclos.

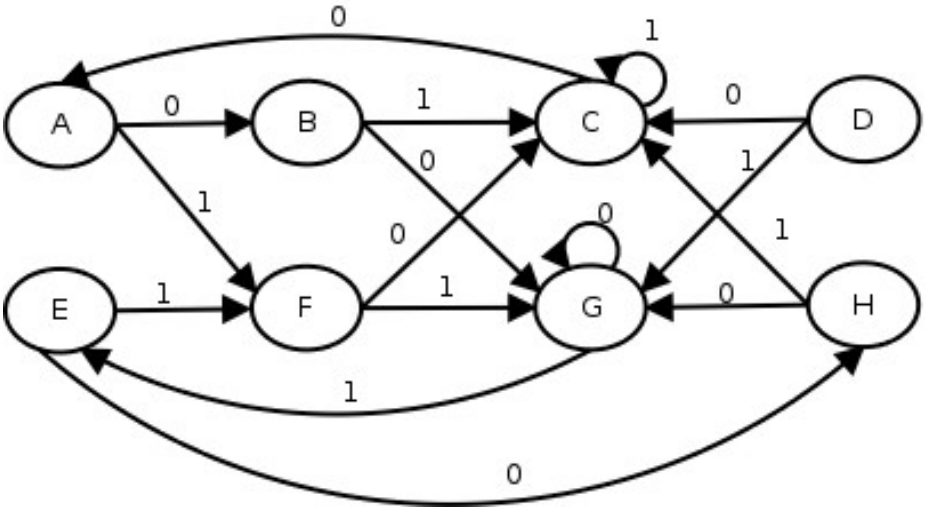
- En este ejemplo no tenemos estados inaccesibles
- Ahora vamos a ver si podemos eliminar estados de error o estados desde los que no se puede llegar a estados finales.
- En este ejemplo solo tenemos un estado final,entonces lo analizamos, si hubiera mas estados finales los analizamos también.
- Desde Q1 llegamos a estado final , entonces no lo eliminamos.

- Desde Q2 llegamos a estado final , entonces no lo eliminamos.
- Desde Q0 llegamos a estado final , entonces no lo eliminamos.
- Ahora comprobamos si quedan ciclos,en el ejemplo vemos que tenemos varios ciclos por tanto es un autómata infinito.

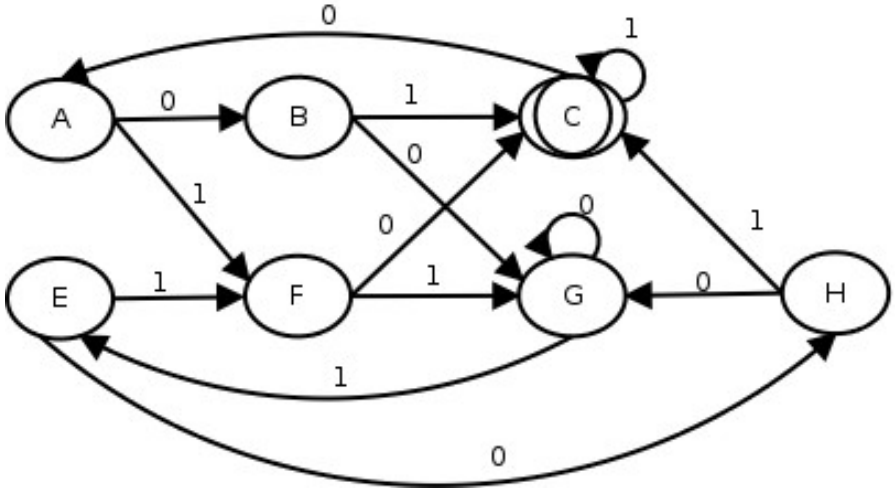
Práctica 7

Autómata minimal

Supongamos el siguiente autómata:



Lo primero que debemos ver es si tiene estado inaccesibles, estos son los que no llega nada a ellos, solo sale, por tanto son inútiles y los eliminaremos. En este caso tenemos el estado D, como vemos no le llega nada. Nuestro nuevo autómata queda así:



El siguiente paso es ver los estados indistinguibles. Estos estados son por decirlo de alguna manera estados que “cumplen la misma función”, al final acabas en el mismo destino, por tanto realizamos el algoritmo de la tabla.

Tacharíamos primero todos los estados finales. En nuestro caso el estado C:

B						
C	X	X				
E			X			
F			X			
G			X			
H			X			
	A	**B**	**C**	**E**	**F**	**G**

Luego vamos apuntando donde cae cada pareja y si va dando en tachado se tacha el que había ya guardado. Finalmente debemos obtener los estados indistinguibles. En nuestro ejemplo la tabla tras ser rellenada queda como sigue:

B	X					
C	X	X				
E		X	X			
F	X	X	X	X		
G	X	X	X	X	X	
H	X		X	X	X	X
	A	**B**	**C**	**E**	**F**	**G**

Si nos fijamos las combinaciones no tachadas son (E,A) y (B,H) por tanto podemos afirmar que son indistinguibles entre ellos, es decir E es indistinguible de A y B es indistinguible de H.

Por tanto ya no es muy difícil pensar como quedará nuestro autómata. Teniendo en cuenta los estados indistinguibles hacemos una especie de “fusión arbitraria”, para verlo mejor pongo como queda el autómata porque no hay mucho mas que explicar: