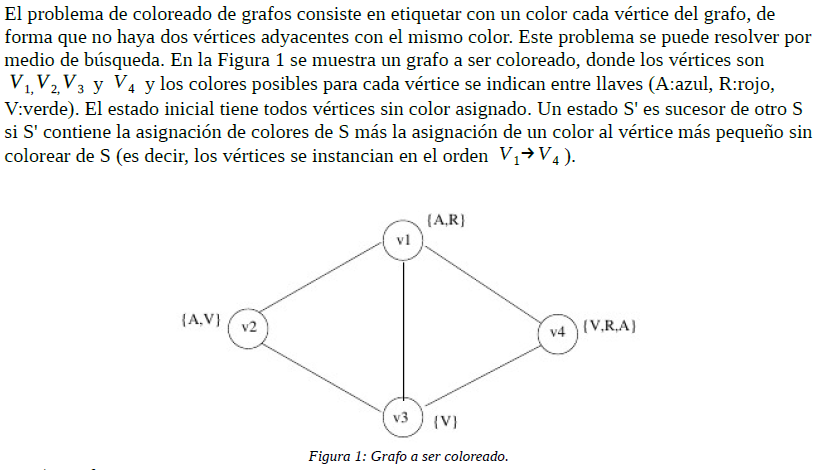
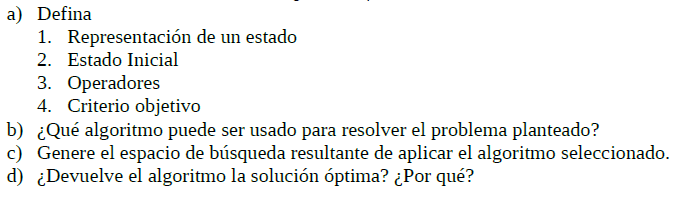
**Soluciones**

Modelar un problema mediante estados





Respuestas:

a) Colores: A: azul, V: verde, R: rojo

Vértices: V1, V2, V3, V4

1. Representación de un estado:

|  |  |  |  |
| --- | --- | --- | --- |
| V1: A | V2: V | V3: A | V4: V |

1. Representación de estado inicial (EI):

|  |  |  |  |
| --- | --- | --- | --- |
| V1: | V2: | V3: A | V4: V |

1. Operadores:

Op1: colorear el V*i* de color azul “A”.

Op2: colorear el V*i* de color verde “V”.

Op3: colorear el V*i* de color rojo “R”.

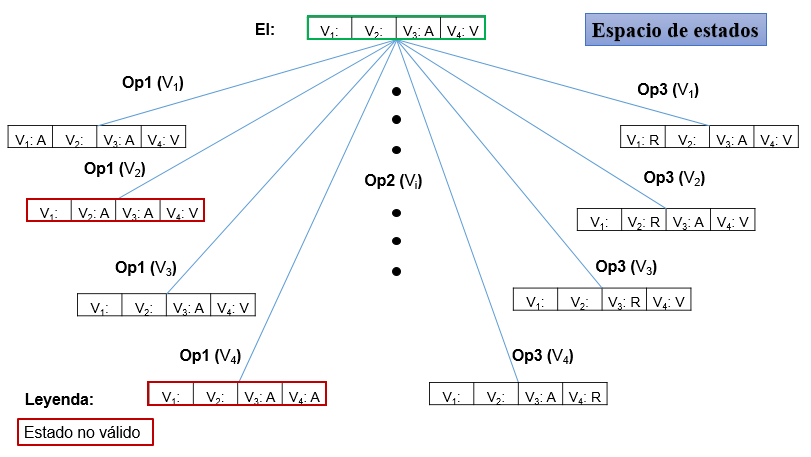
*i*= {1,2,3,4}

1. Criterio Objetivo:

Etiquetar con un color cada vértice (V*i*) del grafo, de forma que no haya dos vértices adyacentes {V***i-1***, V***i***} con el mismo color.

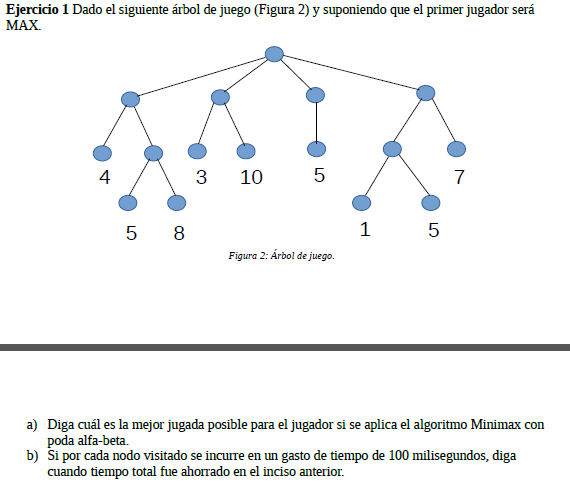
b) R/ El algoritmo que puede ser usado en este problema es Escalador de colinas (*Hill* *Climbing*).

c)



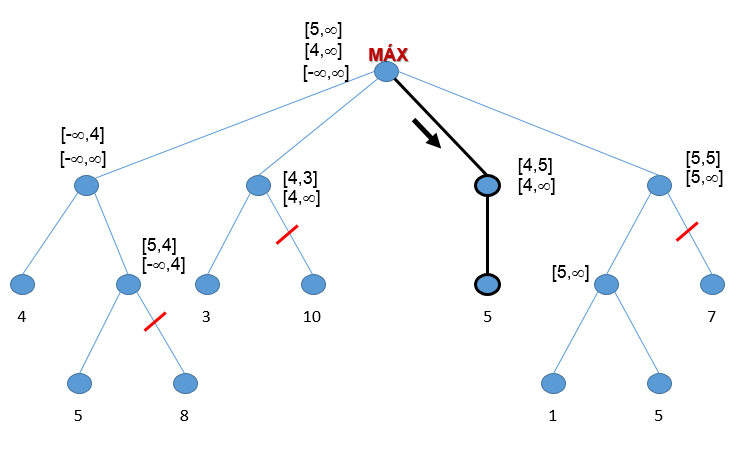
d) El algoritmo no es óptimo, ya que bajo ciertas condiciones se queda con el mejor vecino sin ver más hacia delante. Frecuentemente progresa muy rápido hacia una solución porque le es muy fácil mejorar un mal estado. Por lo anterior es conocido como un *algoritmo voraz local.* En su desarrollo puede detenerse al caer en un máximo local, una meseta o una cresta, por lo que no alcanza el máximo global.

Resolver ejercicios de búsqueda con adversario

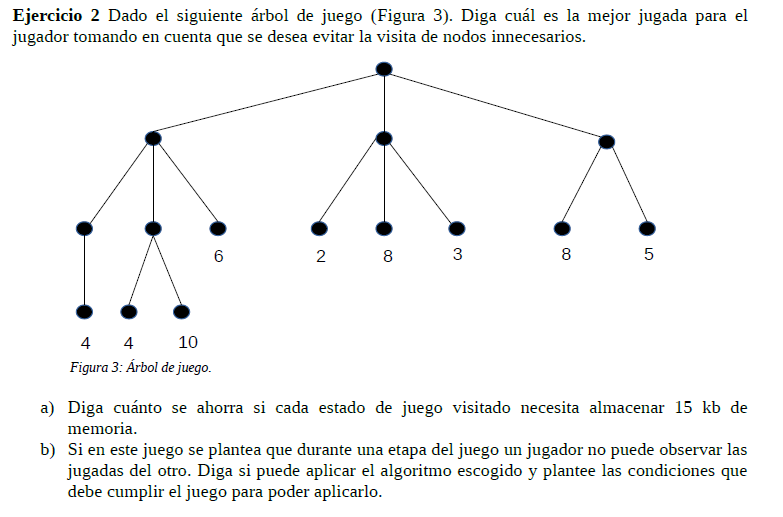


Respuestas:

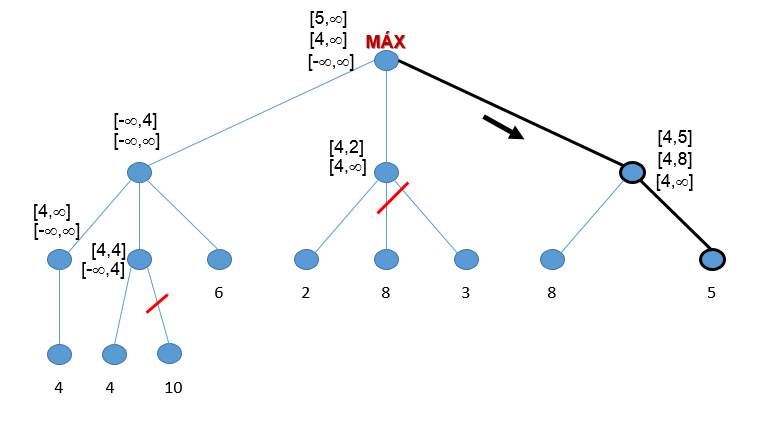
a)



b) Se ahorran 300 ms.

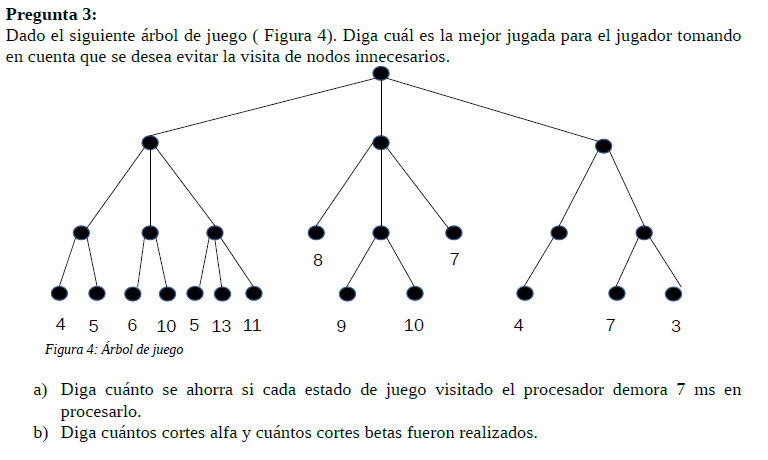


Respuestas:

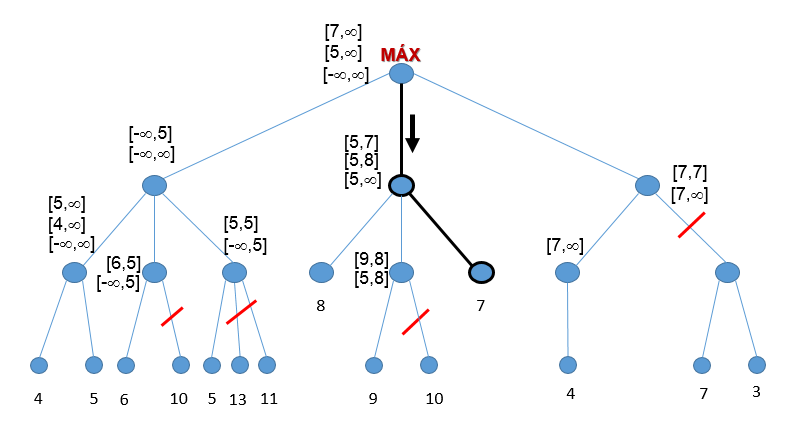
a) 

R/ Se ahorran 45 kb de memoria.

b) No sería posible aplicar el algoritmo pues, el mismo es aplicable a juegos de dos jugadores, con información perfecta: no interviene el azar, cada jugador ve toda la información del otro.



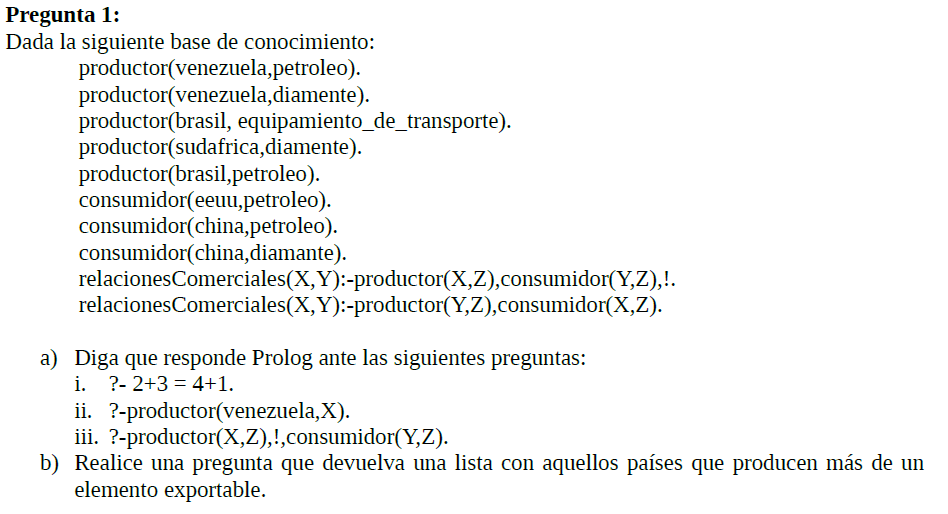
Respuestas:

a) 

R/ Se ahorran 42 ms.

b) R/ Se realizaron 1 corte alpha (α) y 4 cortes beta (β).

Resolver un ejercicio sencillo usando el lenguaje Prolog



Respuestas:

a)

1. false.
2. X = petroleo;

X = diamante.

1. X = venezuela,

Z = petroleo,

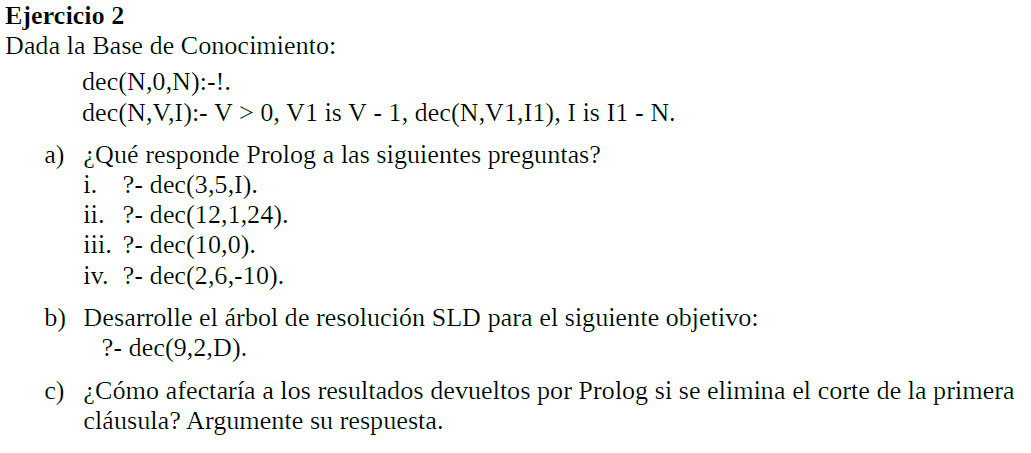
Y = eeuu ;

X = venezuela,

Z = petroleo,

Y = china

b) ?-findall(X, productor(X,\_),LR),findall(Y,(append(\_,[Y|L1],LR),member(Y,L1)),L).



Respuestas:

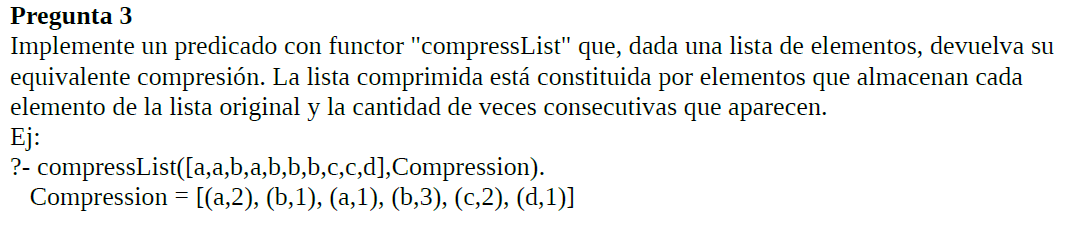
a)

1. I = -12.
2. False.
3. Error (dec/3)
4. True.

b)



c) Si se elimina el predicado corte “!” de la primera cláusula no evitaría que realice el backtracking automático, por lo que Prolog “entiende” que debe realizar una nueva consulta, unificando con la segunda cláusula del predicado. Para la unificación con la última cláusula Prolog respondería “false”.



Respuesta:

compressList(L,Compression):-compressAux(L,LR), findall(E,(member([X,Y],LR), E =.. [(','),X,Y]),Compression).

compressAux([],[]):-!.

compressAux([X|L],[[X,C]|Cant]):- contar([X|L],C,LR),!,compressAux(LR,Cant).

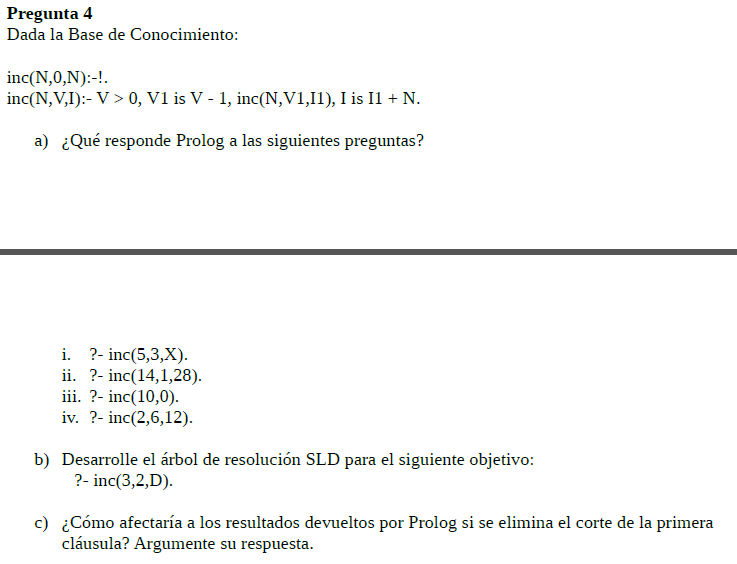
vecesContinuas(\_,[],0):-!.

vecesContinuas(X,[X|L],Cant):- vecesContinuas(X,L,C1),!, Cant is C1 + 1.

vecesContinuas(\_,\_,0).

contar([X|L],C,LR):-vecesContinuas(X,[X|L],C),segmentar(C,[X|L],LR),!.

segmentar(Pos,L,L1):- append(\_,L1,L),length(L,C), Long is C - Pos, length(L1,Long),!.



Respuestas:

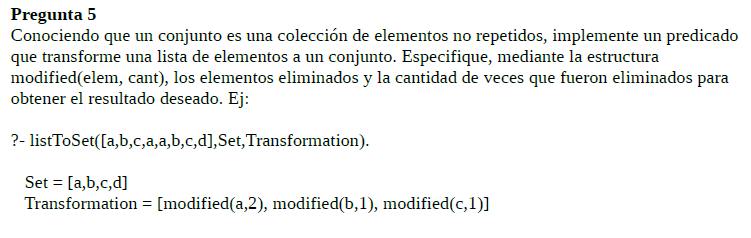
a)

1. X=20.
2. True.
3. Error (inc/3)
4. False.

b)



c) Si se elimina el predicado corte “!” de la primera cláusula no evitaría que realice el backtracking automático, por lo que Prolog “entiende” que debe realizar una nueva consulta, unificando con la segunda cláusula del predicado. Para la unificación con la última cláusula Prolog respondería “false”.



Respuesta:

listToSet(L,Set,Transformation):- listToSetAux(L,LR), findall(E,(member([X,Y],LR), Y\=0, E=..[modified,X,Y]),Transformation), modificarLista(L,Set).

listToSetAux([],[]):-!.

listToSetAux([X|L],[[X,C]|LR]):- eliminarElemento(X,[X|L],L1), length(L,C1), length(L1,C2), C is C1 - C2, !, listToSetAux(L1,LR).

modificarLista([],[]):-!.

modificarLista([X|L],[X|LR]):- eliminarElemento(X,L,L1),!,modificarLista(L1,LR).

eliminarElemento(\_,[],[]):-!.

eliminarElemento(X,[X|L],LR):-!,eliminarElemento(X,L,LR).

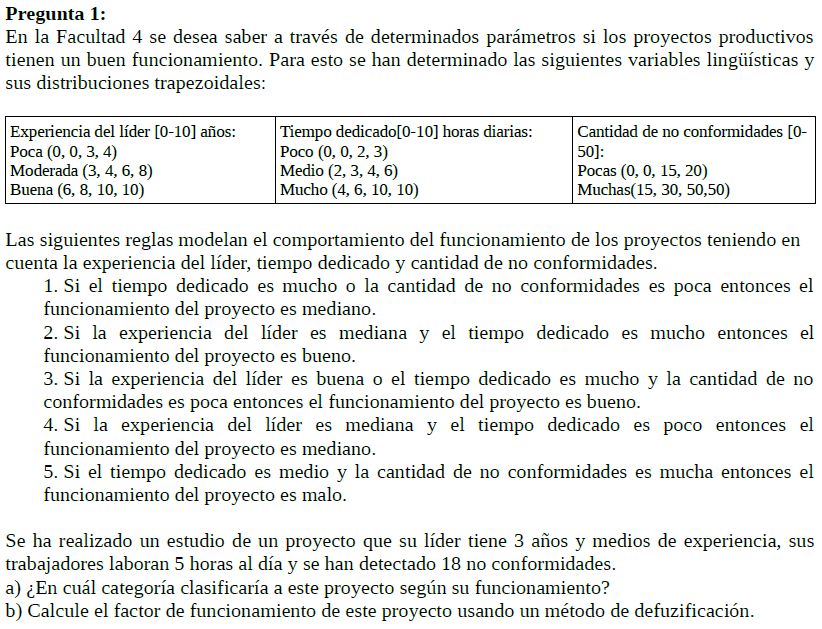
eliminarElemento(X,[Y|L],[Y|LR]):-eliminarElemento(X,L,LR).

cantidadApariciones([],\_,0):-!.

cantidadApariciones([X|L],X,Cant):- cantidadApariciones(L,X,C1),!, Cant is C1 + 1.

cantidadApariciones([\_|L],X,Cant):-cantidadApariciones(L,X,Cant).

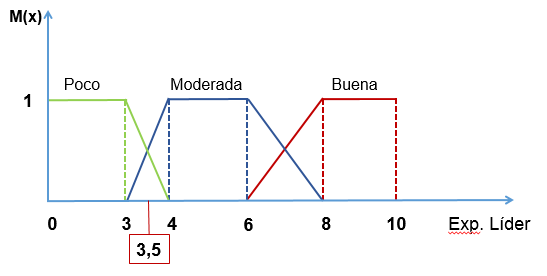
Realizar un ejercicio de ejemplo donde se muestre todo el proceso de trabajo de la lógica difusa.



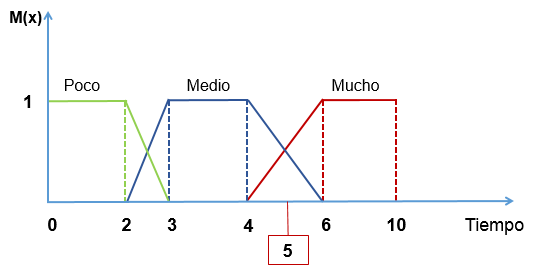
Respuestas:

a) Funciones de pertenencia:

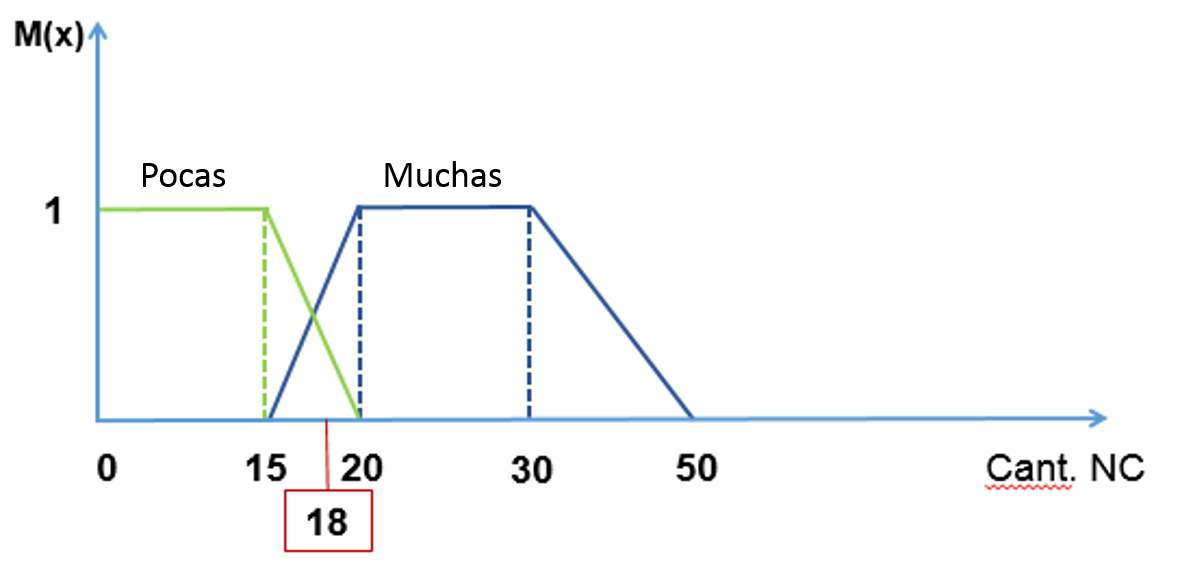
* ***M(X)= (X-A)/(B-A) (Ganma)***
* ***M(X)= (B-X)/(B-A) (L)***



**M**Poco(3,5) = 0,5 **M**Moderada(3,5) = 0,5 **M**Buena(3,5) = 0



**M**Poco(5) = 0 **M**Medio(5) = 0,5 **M**Mucho(5) = 0,5



**M**Pocas(18) = 0,4 **M**Muchas(18) = 0,6

**Utilizando el par mínimo-máximo de las T-Norma y S-Norma.**

|  |  |
| --- | --- |
| **R1**: TM ∨ NCP → FMe FC(1)  **R2**: EM ∧ TM → FB FC(1)  **R3**: EB ∨ TM ∧ NCP → FB FC(1)  **R4**: EM ∧ TP → FMe FC(1)  **R5**: TMe ∧ NCM → FM FC(1)  R1: FC(FMe) = 0,5  R2: FC(FB) = 0,5  R3: FC(FB) = 0,4  R4: FC(FMe) = 0  R5: **FC(FM) = 0,5**  Glob(R1,R4) = **FC(FMe) = 0,5**  Glob(R2,R3) = **FC(FB) = 0,5** | EP: experiencia poca (FC = 0,5)  EM: experiencia moderada (FC = 0,5)  EB: experiencia buena (FC = 0)  TP: tiempo dedicado poco (FC = 0)  TMe: tiempo dedicado medio (FC = 0,5)  TM: tiempo dedicado mucho (FC = 0,5)  NCP: cantidad de NC pocas (FC = 0,4)  NCM: cantidad de NC muchas (FC = 0,6)  FM: funcionamiento del proyecto malo  FMe: funcionamiento del proyecto mediano  FB: funcionamiento del proyecto bueno |

**R/** Según las características analizadas el funcionamiento del proyecto puede clasificarse en cualquiera de las 3 clasificaciones (malo, mediano, bueno), ya que todas tienen el mismo valor de certeza.

b) En este caso se empleó como método de defuzificación la siguiente función:

X = 2\*μ(T malo)+ 3\*μ(T mediano)+ 5\*μ(T bueno)

**X = 5**

**R/** EL funcionamiento del proyecto al tener un líder de 3 años y medio de experiencia, con trabajadores que laboran 5 horas al día y con 18 no conformidades detectadas tiene un valor de 5 puntos (de una escala de 0 a 10).