Pertany un element a una llista ?

```
pertany(X,[X|L]).
pertany(X,[Y|L]):-pertany(X,L).
```

fixau-vos com es pot utilitzar la variable anònima:

```
pertany(X,[X|_]).
pertany(X,[ _ |L]):-pertany(X,L).
```

Pertany X a la llista L ?

?-pertany(2, [1,2,3]).

yes

R1. pertany(X,[X|L]).

R2. pertany(X,[Y|L]):-pertany(X,L).

R1: X=2, X=1, Y=[2,3] no !

R2: X=2, Y=1, L=[2,3] →pertany(2,[2,3]).

R1: X=2, X=2, Y=[3] si !

Que passa si demanam...

?-pertany(X, [1,2,3]).

pertany(X, [1,2,3]).

R1. pertany(X,[X|L]).

R2. pertany(X,[Y|L]):-pertany(X,L).

R1: X=1, X=1, L=[2,3] → escriu X=1 ;

R2: X'=X, Y'=1, L=[2,3] → pertany(X',[2,3]).

R1: X''=X', X'=2, L=[3] → escriu X=2 ;

R2: X''=X',Y=2,L=[3] → pertany(X'',[3]).

R1: X'''=X'', X''=3, L=[] → escriu X=3 ;

R2: X'''=X'',Y=3, L=[] →pertany(X''',[]).

no

Que passa si demanam...

?-pertany(X, [1,2,3]).

X=1;

X=2;

X=3;

no

Afegir dues llistes

```
afegir([],L,L).
afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).

?- afegir([a,b],[c,d],L).
L=[a,b,c,d] ;
no
```

Afegir dues llistes

R1. afegir([],L,L).
R1. afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).
?- afegir([a,b],[c,d],L).

R1: no unifica []=[a,b]
R2: [a,b]=[X|L1], [c,d]=L2, L=[X|L3]
X=a, L1=[b], L2=[c,d], L=[a|L3]
afegir([b],[c,d],L3).

R1: no unifica []=[b]
R2: [b]=[X'|L1'], [c,d]=L2', L3=[X|L3']
X'=b, L1'=[], L2' = [c,d], L3=[b|L3']
afegir([],[c,d],L3')
R1: []=[],L=[c,d],L3'=L → L3'= [c,d]
L3= [b|L3']= [b|[c,d]] = [b,c,d]

L= [a|L3] = [a|[b,c,d]] = [a,b,c,d]

Que passa si demanam...

?-afegir(L1, L2, [a,b,c]).

?- afegir(L1, L2, [a,b,c]).

R1. afegir([],L,L).
R2. afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).

R1: []=L1, L=L2, L=[a,b,c] →**escriu L1=[], L2=[a,b,c];**
R2: [X|L1']=L1, L2'=L2, [X|L3']=[a,b,c]
X=a, L3'=[b,c] → afegir(L1',L2',[b,c])
R1: []=L1',L=L2', L=[b,c] →L1'=[], L2'=[b,c]
L1=[a|L1']=[a], L2=L2'=[b,c] →**escriu L1=[a], L2=[b,c];**
R2:[X|L1'']=L1', L2''=L2', [X|L3'']=[b,c] →X=b,L3''=[c]
afegir(L1'',L2'',[c])
R1: []=L1'',L=L2'', L=[c] →L1''=[], L2''=[c]
L1'=[b|L1'']=[b], L2'=L2''=[c]
L1=[a|L1']=[a,b], L2=L2'= [c] →**escriu L1=[a,b], L2=[c];**
R2: ...

Que passa si demanam...

?-afegir(L1, L2, [a,b,c]).

L1=[]

L2=[a,b,c];

L1=[a]

L2=[b,c];

L1=[a,b]

L2=[c];

L1=[a,b,c]

L2=[];

no

Trobar el darrer element d'una llista

```
darrer([X], X).
darrer([X|L],Y):-darrer(L,Y).
```

fixau-vos com es pot utilitzar la variable anònima:

```
darrer([X], X).
darrer([ _ |L],Y):-darrer(L,Y).
```

Invertir una llista

```
invertir([],[]).
invertir([X|L1],L2):- invertir(L1,L3),
                      afegir(L3,[X],L2).
```

Esborrar un element d'una llista

```
borrar(X,[],[]).
borrar(X,[X|L1],L2):-borrar(X,L1,L2).
borrar(X,[Y|L1],[Y|L2]):-X\=Y,borrar(X,L1,L2).
```

Permutar els elements d'una llista

```
permuta([],[]).
permuta([X|L1],L2):-permuta(L1,L3),
                    inserta(X,L3,L2).

inserta(X,L,[X|L]).
inserta(X,[Y|L1],[Y|L2]):-inserta(X,L1,L2).
```

Predicats de control

En PROLOG existeixen predicats que no ens diuen res del nostre univers de discurs:

- Expressen informació de control
- Permeten la comunicació amb els programadors
- Afecten a l'estructura interna de les proposicions
- Afegeixen o eliminen informació a la base de coneixements

Lectura i escriptura de termes

- write(X): mostra X per la pantalla
- nl: nova línia
- tab(X): desplaça el cursor X espais a la dreta
- read(X): llegeix el següent terme entrat pel teclat (acabat en punt)

Lectura i escriptura a fitxers

- tell(X): defineix el fitxer de treball de sortida
- telling(X): retorna el fitxer de treball de sortida
- told: tanca el fitxer de treball i estableix la sortida a la pantalla
- see(X), seing(X) i seen: és el mateix que abans però per la lectura

Consulta de bases de coneixement

- consult(X): llegeix totes les clàusules del fitxer X.
- reconsult(X): si en el conjunt de noves clàusules en hi ha algunes que ja estan a la base de coneixements, es reemplacen per les noves.

- listing(X): mostra totes les clàusules referents al predicat X. Sense arguments, mostra tot el que hi ha a la base de coneixements

Consulta de bases de coneixement

- El PROLOG permet que l'entrada dels consult(X) i reconsult(X) es simplifiquin en forma de llista:

  ?-[arxiu1, arxiu2, -arxiu3].

  representa el consult dels arxius 1 i 2 i el reconsult de l'arxiu 3.

  ?-[user].

  permet l'entrada interactiva de fets i regles

Inserció i esborrat de clàusules

- asserta(X): afegeix noves clàusules al principi de la base de coneixements
- assertz(X): afegeix noves clàusules al final de la base de coneixements
- retract(X): Elimina la primera clàusula que unifiqui amb X

Operador de tall: el *cut* (!)

- Permet controlar la seqüència d'execució modificant el procés que el PROLOG segueix habitualment
- És un objectiu que sempre es satisfà però que mai pot ser resatisfet
- L'efecte a l'execució és l'esborrat de les marques de *backtracking* de la seva esquerra

Operador de tall: el *cut* (!)

- Utilitat: accelera l'execució d'un programa i possibilita l'estalvi de memòria.
- Inconvenient: una utilització incorrecta pot fer que un programa deixi de funcionar

Operador de tall: el *cut* (!)

Utilització:

1. Com a confirmació de que s'ha trobat la regla correcta per resoldre un objectiu (esquema condicional "si-sino")

a:-b, !, c.	si b llavors c
a:-d.	sino d

Operador de tall: el *cut* (!)

Utilització:

2. Com a advertència de que es va per mal camí (juntament amb el predicat *fail*), que no hi ha solucions alternatives

   Ex: implementació del not

Operador de tall: el *cut* (!)

Implementació del not:

```
not(P):-call(P), !, fail.
not(P).
```

Operador de tall: el *cut* (!)

Utilització:

3. Acabament de la generació de solucions múltiples. No es volen generar més solucions alternatives

   Ex: divisió entera de dos números

Operador de tall: el *cut* (!)

Divisió entera:

```
diventera(A,B,C):-natural(C),Y1 is C*B,
                  Y2 is  (C+1)*B,
                  Y1=<A,
                  Y2>A, !.
natural(0).
natural(X):-natural(Y), X is Y+1.
```

Operador de tall: el *cut* (!)

Utilització incorrecte:

```
afegir([ ],X,X):-!.
afegir([A|B],C,[A|D]):-afegir(B,C,D).
```

Operador de tall: el *cut* (!)

```
a(X):-b(X),c(X).
b(1).
b(4).
c(X):-d(X),!,e(X).
c(X):-f(X).
d(X):-g(X).
d(X):-h(X).
e(3).
f(4).
g(2).
h(1).     ?-a(X).
```

Exemple 1.

Donada una base de coneixements de noms d'animals volem descriure el predicat mutant, que descriu la mutació entre dos animals.

Dos animals muten si el nom d'un d'ells acaba amb les mateixes lletres que l'altre comença.

Exemple 1.

base de coneixements:

```
animal([c,o,c,o,d,r,i,l]).
animal([t,o,r,t,u,g,a]).
animal([l,l,o,p]).
animal([g,o,r,r,i,o]).
animal([o,n,s,o]).
animal([g,a,l,l,i,n,a]).
animal([s,o,m,e,r,a]).
```

Exemple 1.

?- mutant(X).

X=[c,o,c,o,d,r,i,l,l,o,p];

X=[t,o,r,t,u,g,a,l,l,i,n,a];

X=[g,o,r,r,i,o,n,s,o];

X=[o,n,s,o,n,s,o];

X=[o,n,s,o,m,e,r,a];

no

Solució.

```
mutant(X):-
        animal(A), animal(B),
        afegir(CapA,CoaA,A), CapA\=[], CoaA\=[],
        afegir(CapB,CoaB,B),
        CoaA = CapB,
        afegir(CapA,B,X).
```

Exemple 2.

Donada la següent operació matemàtica:

$$\begin{array}{r} T\ R\ E\ S \\ +\ D\ O\ S \\ \hline C\ I\ N\ C \end{array}$$

quins valors numèrics hem d'assignar a cada lletra perquè la suma tengui sentit (lletres diferents tenen valors diferents i la T i la C són distintes de zero).

Solució

```
sum(X, 0,0,X,0).
sum(R, X, Y, Z, R2):-
        residu(R), digit(X), digit(Y),
        T is R+X+Y, R2 is T // 10,
        Z is T mod 10.
```

Solució

```
digit(X):-pertany(X, [0,1,2,3,4,5,6,7,8,9]).
residu(0).
residu (1).
```

Solució

```
suma:-
    sum(0, S,S,C,R1),
    sum(R1,E,O,N,R2),
    sum(R2,R,D,I,R3),
    sum(R3,T,0,C,0),
    T \== 0, C \== 0,
    differs([T,R,E,S,D,O,C,I,N]),
    write([T,R,E,S,D,O,S,C,I,N,C]).
```

Exemple 3.

El Dr. Quimiolvido té sis botelles plenes de líquids de colors (vermell, taronja, groc, verd, blau i lila). Sap que alguns d'aquests líquids són tòxics però no recorda quins... el que sí recorda és que:

A cada una de les següents parelles de botelles, en hi ha una amb verí i l'altra no:

- Les botelles lila i blava
- Les botelles vermella i groga
- Les botelles blava i taronja

A més, el Dr. Quimiolvido també recorda que en les següents parelles en hi ha un sense verí:

- El lila i el grog
- El vermell i el taronja
- el verd i el blau

Exemple 3.

Ah!!!

m'acaba de dir el Dr. que està segur de que el líquid de la botella vermella no és tòxic.

Podeu plantejar un programa en PROLOG que l'ajudi a trobar els colors de les botelles amb verí ?

Exemple 4.

Aquest any han començat a la universitat cinc amics de les trobades d'estiu. Des de principi de curs (excepte els caps de setmana, que tornen a ca seva) viuen a la residència d'estudiants i ocupen habitacions consecutives. Cada un té un cotxe de diferent color i diferent marca i és natural d'un poble distint. Escriviu un predicat de PROLOG que ens respongui a la pregunta:

Exemple 4.

Aquest any han començat a la universitat cinc amics de les trobades d'estiu. Des de principi de curs (excepte els caps de setmana, que tornen a ca seva) viuen a la residència d'estudiants i ocupen habitacions consecutives. Cada un té un cotxe de diferent color i diferent marca i és natural d'un poble distint. Escriviu un predicat de PROLOG que ens respongui a la pregunta:

Qui és de Valldemossa ?

Exemple 4.

- En Pep estudia història
- Na Joana viu al costat de qui és de Binissalem
- En Manel viu al costat de qui és de Palma
- Qui té un cotxe vermell té l'habitació a l'esquerra de qui el té blau
- Qui té el cotxe vermell estudia pedagogia
- Qui té el Fiat viu al costat de qui té el cotxe negre
- Na Joana té l'habitació al costat de qui estudia dret
- El Volkswagen és de color gris
- Qui té un Renault és de Manacor
- Qui té el Fiat esta a la primera habitació
- Na Maria té un Audi
- En Toni és de Sóller
- Qui estudia informàtica té un BMW
- En Manel té un cotxe blanc
- A qui estudia econòmiques li ha tocat l'habitació del mig

TEMA V. Exemples de L.P.A.N.

2721-Llenguatges de Programació

Ramon Mas