

Fórmulas

Predicados básicos de árbol genealógico:

abuelo(X, Y) :- progenitor(X, Z), progenitor(Z, Y).
hermano(X, Y) :- progenitor(Z, X), progenitor(Z, Y), X \= Y.
tio(X, Y) :- progenitor(Z, Y), hermano(Z, X), X \= Y.
primo(X, Y) :- progenitor(Z, X), tio(Z, Y), X \= Y.
tiopolitico(X, Y) :- pareja(X, Z), tio(Z, Y) ; pareja(Z, X), tio(Z, Y), X \= Y.

Saber si una persona es descendiente de otra (es decir, que si X es bisabuelo de Y, o abuelo o padre, o algo así).

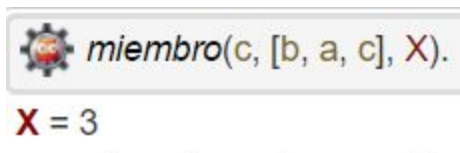
descendiente(X, Y) :- progenitor(X, Y).
descendiente(X, Y) :- progenitor(X, Z), descendiente(Z, Y).

Saber si un elemento es miembro de una lista:

miembro(X, [X|_]).
miembro(X, [_|Z]) :- miembro(X, Z).

Determinar el índice de la posición encontrada de un valor

miembro(X, [X|_], 1).
miembro(X, [_|Z], Cont) :- miembro(X, Z, Cont2), Cont is Cont2 + 1.



Sacar la cabeza de una lista:

elemento1(X, [X|_]).

Sacar el 2do elemento de una lista:

elemento2(X, [_|X|_]).

Sacar el 3er elemento de una lista:

elemento3(X, [_|_|X|_]).

Sacar el 4to elemento de una lista:

elemento4(X, [_|_|_|X|_]).

Concatenación de una lista con otra:

concatenar([], L, L).

concatenar([X|Y], Z, [X|U]) :- concatenar(Y, Z, U).

Factorial de un número:

fact(0, 1).

fact(N, Y) :- M is N-1, fact(M,Z), Y is N*Z.

Otra forma:

fact(X, Y) :- fact_aux(X, 1, Y).

fact_aux(0, Y, Y).

fact_aux(N, Y, Z) :- U is N*Y, M is N-1, fact_aux(M, U, Z).



X = 720

Inversa de una lista:

inversa([], []).

inversa([X|Y], L) :- inversa(Y,R), concatenar(R, [X], L).

Último elemento de una lista:

ultimo([X|[]], X).

ultimo([_|Z], X) :- ultimo(Z, X).

Longitud de una lista

ultimo([X|[]], X, 1).

ultimo([_|Z], X, Cont) :- ultimo(Z, X, Cont2), Cont is Cont2 + 1.



Longitud = 3,

Valor = c

Estar celoso una persona de otra:

estaCeloso(S, Y) :- loves(S, X), loves(Y, X).

Determinar si una lista es igual a otra, pero los dos primeros elementos pueden estar intercambiados.

$\text{swap12}(L1, L2) :- L1 = [\text{First}, \text{Second} | T], L2 = [\text{Second}, \text{First} | T] ; L2 = [\text{First}, \text{Second} | T].$


 `swap12([1, 2, 3], [2, 1, 3]).`

`true`

Repetir todos los elementos de una lista una vez.

$\text{twice}([], []).$

$\text{twice}([H_a | T_a], [H_a, H_a | T_b]) :- \text{twice}(T_a, T_b).$


 `twice([a, b, c], X).`

`X = [a, a, b, b, c, c]`

Sumarle uno a cada elemento de la lista

$\text{addone}([], []).$

$\text{addone}([H_1 | T_1], [H_2 | T_2]) :- H_2 \text{ is } H_1 + 1, \text{addone}(T_1, T_2).$


 `addone([1, 2, 3], X).`

`X = [2, 3, 4]`

Combinar los elementos de dos listas

$\text{combine1}([], [], []).$

$\text{combine1}([H_1 | T_1], [H_2 | T_2], [H_1, H_2 | T_3]) :- \text{combine1}(T_1, T_2, T_3).$


 `combine1([a, b, c], [1, 2, 3], X).`

`X = [a, 1, b, 2, c, 3]`

Combinar en sublistas los elementos de dos listas

$\text{combine2}([], [], []).$

$\text{combine2}([H_1 | T_1], [H_2 | T_2], [[H_1, H_2] | T_3]) :- \text{combine2}(T_1, T_2, T_3).$

 `combine2([a, b, c], [1, 2, 3], X).`

`X = [[a, 1], [b, 2], [c, 3]]`

Combinar los elementos de dos listas en sublistas, concatenandoles una j a cada sublista.

`combine3([], [], []).`

`combine3([H1 | T1], [H2 | T2], [j(H1, H2) | T3]) :- combine3(T1, T2, T3).`

 `combine3([a, b, c], [1, 2, 3], X).`

`X = [j(a, 1), j(b, 2), j(c, 3)]`

Multiplicar todos los elementos de una lista por un valor X.

`scarlarMult(_, [], []).`

`scarlarMult(X, [H1|T1], [H2|T2]) :- H2 is H1*X, scarlarMult(X, T1, T2).`


 `scarlarMult(2, [1, 2, 3], X).`

`X = [2, 4, 6]`

Sumar las multiplicaciones de dos listas en un solo valor

`dot([], [], 0).`

`dot([H1|T1], [H2|T2], S) :- dot(T1, T2, S2), S is S2+(H1*H2).`

 `dot([1, 2, 3], [1, 2, 3], X).`

`X = 14`

Es decir $\rightarrow 1 * 1 + 2 * 2 + 3 * 3 = 1 + 4 + 9 = 14$

precio(X, Y), Y < 1000. \rightarrow Nos dará todos los articulos X, que tengan un precio Y menor que 1000.

precio(X, Y), Y >= 10000, Y <= 40000. \rightarrow Nos dará todos los artículos X que tengan un precio Y entre 10,000 y 40,000. Nota, el menor que se pone después del igual en Prolog.

Saber el porcentaje de juegos ganados por persona.

porcentaje ganancia(X, Y) :- ganados(X, G), jugados(X, J), Y is (G/J)*100. → X es la persona, e Y es el porcentaje ganado, G es la cantidad de juegos ganados y J es la cantidad de juegos jugados.

read(U), X is U*2, write('el numero es'), write(X). → Nos dejará leer un número, lo multiplicará por 2 y lo guardará en X, imprimirá "el numero es" e imprimirá X.

porcentaje ganancia(X, Y) :- ganados(X, G), jugados(X, J), Y is (G/J)*100, write(Y), write('%'). → Nos saca el porcentaje ganado entre las partidas jugadas y ganadas y lo escribe y escribe el simbolo de porcentaje.

Teoría

La **lógica** es una ciencia formal que se basa en la demostración y la inferencia.

Un **argumento** es correcto si su conclusión es consecuencia lógica de sus premisas, de otro modo es incorrecto. Un **argumento** es un sistema de enunciados, que consta de premisas y una conclusión.

Existen dos tipos de **sistemas lógicos**: Lógica proposicional y Lógica de predicados.

La **lógica proposicional** se basa en proposiciones y la de **predicados** se basa en predicados y esta distingue entre sujeto y predicado..

La **lógica de predicados** considera el mundo compuesto por objetos y propiedades o relaciones. Los objetos se conocen como argumentos o terminos y las relaciones o propiedades se conocen como predicados. Algo que hay que notar es que un predicado puede ser verdadero para un argumento pero falso para otro.

Las **constantes lógicas** son: verdadero y falso.

Los **símbolos de constantes** son letras mayusculas.

Los **simbolos de predicados y funciones** son letras minisculas.

Los **simbolos de variables** son x,y, z.. Etc.

Una **oración atomica** esta formada por un predicado y una lista de argumentos entre parentesis. Ej: hermano(oscar, omar).

Los **términos** es una expresión que se refiere a un objeto, es el argumento del predicado. Cuando un termino no tiene variables se conoce como **término de base**.

Funtor es el nombre de una función o predicado y **aridad** el numero de argumentos.

Los **cuantificadores** son universales y existenciales. Universales \rightarrow Para todo x tal., Existenciales \rightarrow Existe un x que tal...

La **programación lógica** se basa en el paradigma de programación declarativa y se basa en el concepto de funciones.

Prolog significa Programming in logic o Programacion Logique. Utiliza la lógica de primer orden o de predicados y fue desarrollado en la Universidad de Marsella (colmarauer) en los 70's.

Las **reglas de prolog** se hacen utilizando cláusulas de Horn y utiliza modus ponens es decir, si es verdad el antecedente es verdad el consecuente. En prolog primero se escribe el consecuente y luego el antecedente.

Los **atomos en prolog** deben comenzar en miniscula y pueden tener espacios por medio si estan escritos entre comillas simples, de otra manera, deben tener rayitas abajo además de espacios, y si comienzan con mayuscula deben estar entre comillas.

Ej: **oscar**, 'Oscar', 'Hola_Oscar', **oscar123**, 'Oscar123', **oSCAR**.

Las **variables** comienzan con letra mayuscula o guion bajo. Ej: **Oscar**, **_oscar**, **X**, **_x**.

Se dice que dos terminos **unifican** si son identicos o las variables de los dos terminos pueden instanciar objetos que pueden llegar a ser identicos.

Una **lista** es una secuencia de elementos, en prolog si tenemos [X|Y], X es la cabeza e Y es el resto de la lista que puede ser otra lista. Una lista vacía en prolog se representa con [], es decir un corchete abierto y uno cerrado juntos.

NOTA: Cualquier teoría que no haya sido incluida aqui es porque esta explicada en las anotaciones de las clases. Sería lo mejor si chequearan las notas actualizadas que he puesto de esta materia, ya que ahí se encuentran los casos de unificación que irán en en el examen.