Predicados - Prolog

Programación declarativa (Universidad Rey Juan Carlos)

PROLOG – PREDICADOS.

UNIFICACIÓN.

UNIFICACIÓN	PROPÓSITO	EJEMPLO
X = Y	Cierto si X e Y son términos unificables. Toda variable se puede unificar con una variable o constante, incluidos los términos compuestos. Una constante sólo se puede unificar con otra constante idéntica.	?- X = pepe. => X = pepe
X \= Y	Cierto si X e Y son términos no unificables.	?- pepe \= pepa. => true

LISTAS.

LISTAS	PROPÓSITO	EJEMPLO
append(?L1, ?L2, ?L)	Cierto si L es la concatenación de las listas L1 y L2.	?- append(L1, L2, [1, 2, 3]). => {L1 = [], L2 = [1, 2, 3]} U {L1 = [1], L2 = [2, 3]} U {L1 = [1, 2], L2 = [3]} U {L1 = [1, 2, 3], L2 = []}
member(?E, ?L)	Cierto si el elemento E pertenece a lista L.	?- member(2, [1, 2, 3]). => true
length(?L, ?N)	Cierto si N es la longitud de la lista L.	?- length([1, 2, 3], N). => N = 3
is_list(+L)	Cierto si L es una lista.	?- is_list([1, 2, 3]). => true
sumlist(+L, ?N)	Cierto si L es una lista de números cuya suma es N.	?- sumlist([1, 2, 3], N). => N = 6
map(+P, +L)	Cierto si L es una lista en la que todos sus elementos cumplen la propiedad	?- map(integer, [1, 2.5, 3]). => false

	P/1.	
map(+P, ?L1, ?L2)	Cierto si P(x_i, y_i) se cumple para todos los pares (x_i, y_i) donde x_i e y_i son los iésimos elementos de L1 y L2 respectivamente.	?- map(doble, [1, 2, 3], L2). => L2 = [2, 4, 6]
include(+P, ?L1, ?L2)	Cierto si la lista L2 está formada por los elementos de L1 que cumplen la propiedad P/1. Es equivalente al predicado filter.	?- include(par, [1, 2, 3, 4, 5], [2, 4]). => true
exclude(+P, ?L1, ?L2)	Cierto si la lista L2 está formada por los elementos de L1 que no cumplen la propiedad P/1.	?- exclude(par, [1, 2, 3, 4, 5], [1, 3, 5]). => true
pliegai(+Obj, +L, +VI, -VF)	Cierto si VF es el resultado (valor final) de plegar mediante Obj la Lista L desde la izquierda partiendo del valor inicial VI.	?- pliegai(suma, [1, 2, 3], 4, VF). => 10
pliegad(+Obj, +L, +VI, -VF)	Cierto si VF es el resultado (valor final) de plegar mediante Obj la Lista L desde la derecha partiendo del valor inicial VI.	?- pliegad(resta, [1, 2, 3], 4, VF). => -2
reverse(?L1, ?L2)	Cierto si L2 es la inversa de la lista L1.	?- reverse([1, 2, 3], L2). => L2 = [3, 2, 1]

RECOLECCIÓN DE SOLUCIONES.

1. Bagof:
   bagof(?T, +Obj, ?L) → Cierto si L es una lista no vacía que contiene todas las instancias de T que cumplen con la cláusula objetivo Obj.
   Características:
   - Permite repeticiones.
   - Devuelve el resultado en orden en el que Prolog se encuentra las soluciones.

- Falla si Obj no es cierto para ninguna instancia de T, es decir, no es capaz de devolver una lista L vacía.
- La cuantificación de las variables es universal, es decir, el predicado proporciona varias soluciones, una por cada posible valor de la variable libre. Para evitar este comportamiento se pueden cuantificar existencialmente mediante el uso del operador ^.

2. Setof:
   setof(?T, +Obj, ?L) → Cierto si L es una lista no vacía que contiene todas las instancias de T que cumplen con la cláusula objetivo Obj.
   Características:
   - No permite repeticiones.
   - Devuelve el resultado en orden ascendente.
   - Falla si Obj no es cierto para ninguna instancia de T, es decir, no es capaz de devolver una lista L vacía.
   - La cuantificación de las variables es universal, es decir, el predicado proporciona varias soluciones, una por cada posible valor de la variable libre. Para evitar este comportamiento se pueden cuantificar existencialmente mediante el uso del operador ^.
3. Findall:
   findall(?T, +Obj, ?L) → Cierto si L es una lista no vacía que contiene todas las instancias de T que cumplen con la cláusula objetivo Obj.
   Características:
   - Permite repeticiones.
   - Devuelve el resultado en orden en el que Prolog se encuentra las soluciones.
   - No falla si Obj no es cierto para ninguna instancia de T, devuelve la ñista Ñ vacía.
   - La cuantificación de las variables es existencial, es decir, la única solución de la ejecución del predicado es la lista L.

CLASIFICACIÓN DE TÉRMINOS.

CLASIFICACIÓN DE TERMINOS	PROPÓSITO	EJEMPLOS
atomic(+T)	Cierto si T es una constante (número o átomo).	?- atomic(X). => false
number(+T)	Cierto si T es un número.	?- number(12). => true
integer(+T)	Cierto si T es un número entero.	?- integer(2.5). => false
natural(+T)	Cierto si T es un número natural, es decir, mayor o igual a cero. No existe en Prolog.	?- natural(-1). => false
float(+T)	Cierto si T es un número real.	?- float(2.5). => true
atom(+T)	Cierto si T es un átomo.	?- atom(pepe). => true
var(+T)	Cierto si T es un término variable.	?- var(X). => true

nonvar(+T)	Cierto si T no es un término variable.	?- nonvar(X). => false
compound(+T)	Cierto si T es un término compuesto.	?- compound(progenitor(X)). => true

COMPARACIÓN DE TÉRMINOS.

COMPARACIÓN DE TÉRMINOS	PROPÓSITO	EJEMPLOS
T1 == T2	Cierto si T1 y T2 son literalmente idénticos, es decir, son la misma constante, la misma variable (con el mismo nombre) o dos términos compuestos literalmente idénticos, concretamente, el mismo functor (nombre), misma aridad y argumentos idénticos.	?- X = Y, X == Y. => true
T1 \== T2	Cierto si T1 y T2 no son literalmente idénticos.	?- progenitor(pepe, X) \== progenitor(pepe, Y). => true

ARITMÉTICOS.

ARITMÉTICOS	PROPÓSITO	EJEMPLOS
X is Y	Si Y es una expresión aritmética, se evalúa y el resultado se intenta unificar con X. Su uso puede dar lugar a error cuando la parte derecha no es una expresión aritmética como, por ejemplo, en ?- X is a + 1. También puede dar lugar a error cuando la parte derecha es una expresión aritmética pero no se puede evaluar como, por ejemplo, en ?- X is 4 * Z.	?- X is sqrt(4), Y is X + 1. => X = 2.0, Y = 3.0
X =:= Y	Cierto si los valores numéricos de X e Y son iguales. Da error si alguno de los	?- 3 + 5 =:= 8. => true

	argumentos no es una expresión aritmética o no se puede evaluar.	
X =\= Y	Cierto si los valores numéricos de X e Y son diferentes. Da error si alguno de los argumentos no es una expresión aritmética o no se puede evaluar.	?- 2 =\= 3 + 4. => true
X < Y	Cierto si el valor numérico de X es menor que el de Y. Da error si alguno de los argumentos no es una expresión aritmética o no se puede evaluar.	?- 2 < 1. => false
X =< Y	Cierto si el valor numérico de X es menor o igual que el de Y. Da error si alguno de los argumentos no es una expresión aritmética o no se puede evaluar.	?- 2 <= 3. => true
X > Y	Cierto si el valor numérico de X es mayor que el de Y. Da error si alguno de los argumentos no es una expresión aritmética o no se puede evaluar.	?- X + 3 > 7. => ERROR
X >= Y	Cierto si el valor numérico de X es mayor o igual que el de Y. Da error si alguno de los argumentos no es una expresión aritmética o no se puede evaluar.	?- 7 >= 2 + 4 => true