Tema 4. OTROS ASPECTOS de PROLOG

4.1. Aritmética en PROLOG

4.2. Predicados metalógicos

4.3. Entrada / Salida

TEMA 4. OTROS ASPECTOS de PROLOG

Este tema se divide en tres secciones que estudian predicados predefinidos en PROLOG: predicados aritméticos, predicados de manipulación de la base de conocimiento y predicados para la Entrada/Salida. Todos ellos son usuales en los distintos sistemas Prolog que existen en el mercado, aunque podrían variar ligeramente de un sistema a otro.

4.1 Aritmética en PROLOG

En PROLOG la aritmética se realiza con ciertos predicados predefinidos que toman como argumento una expresión aritmética (trabajando sobre enteros y reales) y la evalúa.

Expresiones aritméticas

Una expresión aritmética es un término construido con números, variables y funtores que representan funciones aritméticas. Sólo se permiten ciertos funtores en estas expresiones. Algunos de ellos son los siguientes:

E1+ E2	suma
E1 - E2	diferencia
E1* E2	producto
E1 / E2	cociente
E1 // E2	cociente entero
-E	opuesto
E1 mod E2	módulo

Una expresión aritmética sólo puede ser evaluada si no contiene variables libres. En otro caso aparece un error de evaluación

Predicados aritméticos evaluables

Las expresiones aritméticas sólo son evaluadas cuando se pasan como argumentos a predicados aritméticos evaluables:

X is E Evalúa E y unifica el resultado con X.

Ejemplos:

? X is 4/2 + 3/7.	? X is 2*4, 2 is X//3.	? X is 3, Y is X+4.	? Y is X+4, X is 3.	? 3+4 is 3+4.
X = 2.42857	X = 8	X = 3, Y = 7	ERROR	no

Notas:

1. **"X is X+1"** da FRACASO si X está instanciada en la llamada y ERROR aritmético si X está libre.
2. El orden de los literales es relevante en el uso de predicados evaluables. En el cuarto ejemplo se tiene ERROR porque X está libre.

Comparación aritmética

Los siguientes predicados evalúan sus **dos** argumentos como **expresiones aritméticas** y comparan los resultados.

E1 op E2 Se evalúan las expresiones E1 y E2 y se comparan según los operadores siguientes:

$E_1 > E_2$	mayor
$E_1 >= E_2$	mayor o igual
$E_1 < E_2$	menor
$E_1 =< E_2$	menor o igual
$E_1 =:= E_2$	igual
$E_1 =\= E_2$	distinto

4.2 Predicados metalógicos

A continuación se muestran algunos predicados meta-lógicos usuales en los sistemas PROLOG.

Manipulación de términos

Los siguientes predicados meta-lógicos se usan para examinar el estado de instanciación actual de un término:

var(Term)	Es cierto si Term es una variable libre (no instanciada).
nonvar(Term)	Es cierto si Term no es una variable libre.
ground(Term)	Es cierto si Term no contiene variables libres.
atom(Term)	Es cierto si Term está instanciado a un identificador Prolog *[Nota: no confundir con átomo lógico]*.
atomic(Term)	Es cierto si Term está instanciado a un identificador ó a un número.
number(Term)	Es cierto si Term está instanciado a un número (entero o real).
integer(Term)	Es cierto si Term está instanciado a un entero.
float(Term)	Es cierto si Term está instanciado a un real.
compound(Term)	Es cierto si Term está instanciado a una estructura (es decir, a un término compuesto).

is_list(Term)	Es cierto si Term está instanciado a una lista (es decir, a la lista vacía [] ó a un término con funtor '.' y aridad 2, donde el segundo argumento es una lista.
	Este predicado actúa como si estuviera definido de la siguiente forma:
	`is_list(X) :- var(X), !, fail.`
	`is_list([]).`
	`is_list([_|T]) :- is_list(T).`

Igualdad de términos

La igualdad de términos puede determinarse de diferentes formas. La diferencia clave estriba en si tiene lugar o no la unificación de variables en los términos. El operador "=" es la propia unificación. Esto es, se unifican las variables de los términos que se comparan. Sin embargo, el operador "==" no unifica las variables de los términos que se comparan. Por tanto, una variable (no ligada) sólo será igual a sí misma.

T1 = T2	Es cierto si T1 y T2 pueden unificarse.
T1 \= T2	Es cierto si T1 y T2 no pueden unificarse.
T1 == T2	Es cierto si T1 y T2 son idénticos, es decir, se unifican sin que haya ligaduras de variables.
T1 \== T2	Es cierto si T1 y T2 no son idénticos.

Ejemplos:

? a = X. **X = a** **si**	**? X = Y.** **X = _G3, Y = _G3** **si**	**? a == a.** **si**	**? a == X.** **no**	**? X == Y.** **no**	**? X == X.** **X = _G2** **si**

Acceso a estructuras

a) Conversión de una estructura a lista (y viceversa):

E =.. L **L** es la lista formada por el funtor y los componentes de la estructura **E**.
(un átomo se comporta como un funtor con 0 componentes)

Ejemplos:

? libro(autor, título) =.. L. L = [libro, autor, título]	? E =.. [libro, autor, título]. E = libro(autor, título)	? libro =.. L. L = [libro]	? E =.. [libro]. E = libro

b) Acceso al constructor y a los componentes:

functor(E, F, A) La estructura **E** tiene funtor **F** y aridad **A**

Dos modos de uso de este predicado: **(in, out, out)** y **(out, in, in)**

Ejemplos:

? functor(libro(autor, titulo), N, A).	? functor(X, libro, 2).	? functor(p, N, A).	? functor(X, p, 0).
N = libro, A = 2	X = libro(_G358, _G359)	N = p, A = 0	X = p

arg(P, E, C) La estructura **E** tiene la componente **C** en la posición **P** (contando desde 1). Modos de uso: **(in, in, out)** y **(out,in,in)**

Ejemplos:

? arg(2, libro(autor, titulo), X).	? arg(N, libro(autor, titulo), autor).
X = titulo	N = 1

Gestión de la Base de Cláusulas

Veremos a continuación los principales predicados del sistema PROLOG para gestionar la Base de Cláusulas (BC). Es decir, meta-predicados de consulta, inserción, eliminación, ... de las sentencias (o cláusulas) de programa cargadas en memoria.

Para ilustrar cómo funcionan dichos predicados, veremos ejemplos de ellos sobre la siguiente BC:

```
:- dynamic padre/2, hijo_a/2, abuelo/2.
padre(jon, carlos).
padre(carlos, maria).
hijo_a(X,Y) :- padre(Y, X).
abuelo(X, Z) :- padre(X, Y), padre(Y, Z).
```

NOTA: dynamic +Name/+Arity, ... *informa al intérprete que la definición del(os) predicado(s) puede cambiar durante la ejecución (usando assert y/o retract). En otro caso se consideran estáticos, sin permitir cambios durante la ejecución.*

• Predicado para consultar la BC:

clause(Cabeza,Cuerpo) unifica **Cabeza** (que no debe ser una variable libre) y **Cuerpo** con la cabeza y el cuerpo (respectivamente) de una cláusula de la BC.

Ejemplos:

? clause(padre(X,Y),Z). X = jon, Y = carlos, Z = true ; X = carlos, Y = maria, Z = true	? clause(hijo_a(X,Y),Z). X =_G1, Y =_G2, Z = padre(_G2,_G1)	? clause(abuelo(X,Z),U). X=_G1, Z=_G2, U=padre(_G1,_G3) ',' padre(_G3,_G2)

• Predicados para insertar cláusulas en la BC:

assert(Clausula) ó **assertz(Clausula)** inserta **Clausula** (que debe estar instanciada) en la BC al final de la lista de su procedimiento.

Ejemplo: ? assert((hijo_a(X,Y) :- madre(Y,X))).
Efecto: inserta dicha cláusula como la última del procedimiento "hijo_a".

asserta(Clausula) inserta **Clausula** (que debe estar instanciada a una cláusula) en la BC como la primera de su procedimiento.

Ejemplo: ? asserta(padre(juan,eva)).
Efecto: inserta dicha cláusula delante de las dos del procedimiento "padre".

• Predicados para eliminar cláusulas de la BC:

retract(Clausula) elimina de la BC la primera cláusula unificable con **Clausula** (que no debe ser una variable libre).

Ejemplo: ? retract(padre(carlos,Y))
Y = maria *Efecto:* elimina "padre(carlos,maria)" de la BC.

abolish(Nombre / Aridad) elimina de la BC todas las cláusulas del procedimiento de nombre **Nombre** y aridad **Aridad**.

Ejemplo: ? abolish(hijo_a / 2).
si *Efecto:* elimina de la BC todas las cláusulas del predicado "hijo_a" de aridad 2.

• Predicados para listar las cláusulas de la BC:

listing. Lista todas las cláusulas de la BC.
listing(Nombre). Lista todas las que definen al predicado de nombre **Nombre**
listing(Nombre / Aridad). Lista todas las que definen al predicado de nombre **Nombre** y aridad **Aridad**

***NOTA**: Usad estos predicados para comprobar cómo queda la BC tras hacer assert , retract y abolish en los ejemplos anteriores.*

Los predicados de añadir o eliminar cláusulas permiten programar con efectos laterales, por lo que pueden ser peligrosos. Conviene usarlos de forma "legítima", por ejemplo:

1.- Añadiendo hechos que se deduzcan del programa (lemas).

Ejemplo:

lema(P) :- P, asserta((P :- !)).

P es una metavariable. El corte impide que se llame a otro procedimiento para resolver P. Sólo es legítimo si P es determinista.

2.- Simulación de variables globales.

Ejemplo:

```
asig_var(Nombre,Valor)  :- nonvar(Nombre), retract(valor(Nombre,Valor1)),!, asserta(valor(Nombre,Valor)).
asig_var(Nombre,Valor)  :- nonvar(Nombre), asserta(valor(Nombre,Valor)).
```

Supongamos que representamos nuestras variables globales como términos v(N). Veamos la siguiente sesión:

? asig_var(v(5),10).
si — *Efecto*: inserta en la BC la cláusula "valor(v(5),10)".

? valor(v(5),X).
X = 10

? valor(v(N),X), Y is X+1, asig_var(v(N),Y), valor(v(N),Z).
N = 5, X = 10, Y = 11, Z =11 — *Efecto*: elimina de la BC la cláusula "valor(v(5),10)" e inserta "valor(v(5),11)".

Predicados usuales en PROLOG para recolección de soluciones

findall(Instance, Goal, List) — **List** se unifica con la lista de todas las instancias de **Instance** que hacen cierto a **Goal.**
Si **Goa**l no es cierto para ningún valor de **Instance**, entonces **List** se unifica con la lista vacía [].

Ejemplos:

a(1). a(2). a(3). ?- findall(X, a(X), L). L = [1, 2, 3]	fact(bird, duck). fact(sound, quack). fact(color, white). ?- findall(Nombre:Valor, fact(Nombre, Valor), Lista). Lista = [bird:duck, sound:quack, color:white]

bagof(Instance, Goal, List)

Similar a **findall** excepto en cómo trata las variables que aparecen en **Goal** y no en **Instance** (conocidas como *variables libres*). **Bagof** hace backtracking y produce una lista **List** para cada posible ligadura de las variables libres. Se puede convertir una variable libre a no-libre usando ^ . Si **Goa**l no es cierto para ningún valor de **Instance**, entonces **List** se unifica con la lista vacía [].

Ejemplo:

```
pide(fred, cerveza).
pide(tom, vino).
pide(jane, cerveza).
pide(jane, cola).
```

```
?- bagof(P, pide(X, P), L).          %%  X es libre, por lo que se hará backtracking.
X = fred
L = [cerveza] ;
X = tom
L = [vino];
X = jane
L = [cerveza, cola]

?- bagof(P, X ^ pide(X, P), L).      %%  X es no-libre.
L = [cerveza, vino, cerveza, cola].
```

setof(Instance, Goal, List)

Similar a **bagof** salvo en que **List** está ordenada (según el orden estándar) y sin repetidos.

```
?- setof(P, X ^ pide(X, P), L).      %%  X es no-libre.
L = [cerveza, cola, vino].
```

4.3 Entrada / Salida

Los sistemas PROLOG poseen predicados predefinidos para la entrada y salida. Se verán aquí ejemplos de algunos de ellos.

- **Predicados para la E / S de términos** (sobre el "stream" por defecto)**:**

read(X)	lee un término y lo unifica con **X** (el término debe terminar en punto).
write(T)	escribe el término **T** (en particular, **write('texto')** para escribir texto).
display(T)	escribe el término **T** (sin expandir los operadores)

Ejemplos:

```
?- read(U).
| p(1,2).   %Introducido por usuario
U = p(1, 2)
```

```
?- write(7+4).
7 + 4
```

```
?- display(7+4).
+(7, 4)
```

```
?- read(U), X is U*2, write('el número es '), write(X).
| 6.   %Introducido por el usuario
el número es 12
U = 6
X = 12
```

Para especificar la lectura y escritura de términos sobre un "stream" identificado con **ID** se usará: **read(ID,X)** y **write(ID,X)**

- **Predicados para la E / S de caracteres** (sobre el "stream" por defecto)**:**

get0(X)	lee un carácter y unifica su código con **X**.
get(X)	análogo, pero salta los caracteres no imprimibles (por ejemplo, los blancos)
skip(Char)	lee hasta encontrar el carácter **Char**. Una llamada a **get0(X)** detrás leerá el siguiente carácter a **Char**.
put(Char)	escribe el carácter **Char** (o lo ejecuta si no es imprimible).
nl	produce una nueva línea.
tab(N)	escribe **N** espacios en blanco.

Equivalentemente sobre un "stream" identificado con **ID**: **get0(ID,X) , get(ID, X) , skip(ID,Char) , put(ID,Char) , nl(ID)** y **tab(ID,N)**

NOTA**: **Char** puede ser también el código de un carácter en **skip** y **put**. Por ejemplo: **put(a)** ó **put(97)** escriben ambos el carácter **a.

Ejemplos:

```
?- get0(U).
| a     % Introducido por el usuario
U = 97
```

```
?- put(a), nl,  put(b).
a
b
```

```
? skip(a), get0(T), put(T).
| olas      % Introducido por el usuario
 s
T = 115
```

- **Predicados para construir y analizar átomos:**

  name(Constante, LisCod) establece la relación entre **Constante** (que debe cumplir **atom** ó **integer**) y **LisCod** consistente en la lista de códigos (o string equivalente) de los caracteres que conforman **Constante.**

  atom_codes(Atomo, LisCod) similar al anterior pero **Atomo** debe cumplir **atom** (no integer)

Ejemplos:

```
?- name(hola, T).
  T = [104, 111, 108, 97]
```

```
?-  name(C, "hola").
   C = hola
```

```
?- "hola" == [104,111,108,97]
   si
```

```
?- name(N,"23"), Y is N+5.
   N = 23
   Y = 28
```

- **Predicados para abrir y cerrar ficheros:**

  open(file, mode, ID) abre el fichero de nombre **file,** en modo **mode** (read, write, append,...), con identificador (var) **ID**

  open(file, mode, ID, options) similar al anterior pero describiendo una lista de opciones(alias, type,...)

  close(id) cierra el "stream" de identificador **id**

- **Predicados para modificar los dispositivos de E /S :**

see(fich)	hace que **fich** sea el fichero actual de entrada
seeing(F)	indica en **F** cuál es el dispositivo actual de entrada.
seen	cierra el dispositivo actual de entrada (que volverá a ser **user_input**)
tell(fich)	hace que **fich** sea el fichero actual de salida
telling(F)	indica en **F** cuál es el dispositivo actual de salida.
told	cierra el dispositivo actual de salida (que volverá a ser **user_output**)

EJEMPLO:

Se muestra el uso de algunos predicados de E / S en el siguiente programa que lee frases introducidas por el usuario sobre el terminal.

Programa:

% Programa que lee frases (tecleadas en el terminal). La llamada principal es **leer(X).**

```
% LEER, lee una frase.
% Modo de uso: leer(out frase).

leer([P|Ps]) :- get0(C), leepalabra(C,P,C1), restofrase(P,C1,Ps).

% RESTOFRASE, dada una palabra y el carácter que la sigue, devuelve el resto de la frase.
% Modo de uso: restofrase(in palabra, in caracter, out frase)

restofrase(P, _ , []) :- ultimapalabra(P), !.
restofrase(P, C, [P1|Ps]) :- leepalabra(C,P1,C1), restofrase(P1,C1,Ps).
```

```
% LEEPALABRA, dado un carácter inicial, devuelve una palabra y el carácter que viene detrás de la palabra.
% Modo de uso: leepalabra(in carácter, out palabra, out carácter).

leepalabra(C,P,C1) :- caracter_unico(C), !, name(P,[C]), get0(C1).
leepalabra(C,P,C2) :- en_palabra(C,NuevoC), !, get0(C1), restopalabra(C1,Cs,C2), name(P, [NuevoC | Cs]).
leepalabra(C,P,C2) :- get0(C1), leepalabra(C1,P,C2).


% RESTOPALABRA, dado un carácter inicial, devuelve la lista de caracteres del resto de la palabra
% y el carácter que viene detrás de la palabra.
% Modo de uso: leepalabra(in carácter, out lista_de_caracteres, out caracter).

restopalabra(C, [NuevoC | Cs], C2) :- en_palabra(C,NuevoC), !, get0(C1), restopalabra(C1,Cs,C2).
restopalabra(C,[ ],C).


% CARACTER_UNICO, los siguientes caracteres forman palabra por si mismos.
% Modo de uso: caracter_unico(in código_caracter).

caracter_unico(44).     % ,
caracter_unico(59).     % ;
caracter_unico(58).     % :
caracter_unico(63).     % ?
caracter_unico(33).     % !
caracter_unico(46).     % .
```

```
% ULTIMAPALABRA, las siguientes palabras terminan una frase.
% Modo de uso: ultimapalabra(in palabra).

ultimapalabra('.').
ultimapalabra('!').
ultimapalabra('?').

% EN_PALABRA, trata los caracteres que pueden aparecer dentro de una palabra.
% La segunda cláusula convierte mayúsculas a minúsculas.
% Modo de uso: en_palabra(in caracter, out caracter).

en_palabra(C,C)   :- C>96,  C<123.          %  a b ... z
en_palabra(C,L)   :- C>64,  C<91, L is C+32. %  A B ... Z
en_palabra(C,C)   :- C>47,  C<58.           %  0 1 2 ... 9
en_palabra(39,39).                          %  '
en_palabra(45,45).                          %  -
```

Ejemplo de ejecución:

```
?- leer(X), length(X,N).
|: Esta es una prueba, contaremos
|: las palabras luego.

X = [esta, es, una, prueba, (','), contaremos, las, palabras, luego, '.']
N = 10
```