Predicados extralógicos

Objetivos

- Introducir los predicados de segundo orden
- Introducir los predicados de manipulación de la base de cláusulas y los predicados NB
- Introducir los predicados de Entrada/Salida

Predicados de segundo orden

- Se refieren a conjuntos y a sus propiedades.
- Puede verse como “predicados que invocan otros predicados”:
  - **`call(X)`** es de 2do orden.
- Existe una familia de predicados que permite construir el conjunto de todas las soluciones de un predicado dado.

Predicados de segundo orden

`findall(+Template,+Goal,-Bag)` ← *Bag* es la lista con todas las instancias de *Template* para las cuales se satisface *Goal*. En caso de que *Goal* sea insatisfactible, *Bag* unifica con la lista vacía.

`bagof(+Template,+Goal,-Bag)` ← *Bag* es la lista con todas las instancias de *Template* para las cuales se satisface *Goal* para una "asignación" de las variables que NO aparecen en *Template*. En caso de que *Goal* no sea satisfactible, falla.

`setof(+Template,+Goal,-Bag)` ← Ídem a **`bagof`**, pero con *Bag* ordenada y sin repetidos.

Predicados de segundo orden

```
ancestro(juan,jose).
ancestro(juan,ana).
ancestro(jose,pedro).
ancestro(laura,ana).
ancestro(lucia,jose).

?- findall(X,ancestro(X,Y),L).
L=[juan, juan, jose, laura, lucia].

?- findall(X,ancestrto(X,Y),[juan|Z]).
Z = [juan, jose, laura, lucia].

?- findall(X,ancestro(X,juan),L).
L = [].

?- findall(a(X), ancestro(Y, X),L).
L = [a(jose), a(ana), a(pedro), a(ana), a(jose)].
```

Predicados de segundo orden

```
ancestro(juan,jose).
ancestro(juan,ana).
ancestro(jose,pedro).
ancestro(laura,ana).
ancestro(lucia,jose).

?- bagof(X,ancestro(Y,X),L).
Y = jose,
L = [pedro];
Y = juan,
L = [jose, ana];
Y = laura,
L = [ana];
Y = lucia,
L = [jose].
```

Predicados de segundo orden

```
ancestro(juan,jose).
ancestro(juan,ana).
ancestro(jose,pedro).
ancestro(laura,ana).
ancestro(lucia,jose).

?- bagof(X,ancestro(X,angel), L).
false.

?- findall(X,ancestro(X,angel), L).
L = [].
```

Predicados de segundo orden

```
ancestro(juan,jose).
ancestro(juan,ana).
ancestro(jose,pedro).
ancestro(laura,ana).
ancestro(lucia,jose).

?- setof(X,ancestro(X,Y),L).
Y = ana,
L = [juan, laura];
Y = jose,
L = [juan, lucia];
Y = pedro,
L = [jose].

?- setof(X,ancestro(X,pepe),L).
false.
```

```
?- bagof(ancestro(X,Y),L).
Y = ana,
L = [juan, juan, laura];
Y = jose,
L = [juan, lucia];
Y = pedro,
L = [jose].
```

Predicados de segundo orden

Utilizando predicados de segundo orden, implemente el siguiente predicado:

`interseccion(C1,C2,C3)` ← C3 es la intersección de los conjuntos C1 y C2.

Manipulación de programas

Predicados para:

- Acceder a las cláusulas de un programa (clause), lo vimos la clase anterior
- Agregar cláusulas (assert).
- Eliminar cláusulas (retract).

Manipulación de programas

```
clause(+Head, ?Body)
```

Ejemplo:

Si tengo el predicado member:

```
member(X, [X|Xs]).
member(X, [Y|Ys]) :- member(X, Ys).
```

Sucede lo siguiente:

```
?- clause(member(X,Ys),Body).
Ys = [X|Xs],
Body = true;
Ys = [Y|Xs1],
Body = member(X,Ys1).
```

Manipulación de programas

`assert(+Clause)`

Agrega una cláusula al programa:

```
assert(padre(juan,manuel)).
assert( (padre(X,Y):-hijo(Y,X)) ).
```

`assertz` ← los agrega en el último lugar

`asserta` ← los agrega en el primer lugar

`assert` es equivalente a **`assertz`**

Manipulación de programas

`retract(+Term)`

Elimina una cláusula del programa:

```
retract(padre(_,_)).
retract(padre(_,_):-_).
```

`retractall(X)` ← elimina todas las cláusulas cuyo cabezal unifique con X

Ejemplo: assert

```
assert_test(0):-fail.
assert_test(N):-
    N > 0,
    assert(test(N)),
    N1 is N - 1,
    findall(X,test(X),L),
    writeln(L),
    assert_test(N1).
```

```
?- assert_test(3).
 [3]
 [3,2]
 [3,2,1]
 false.
?-assert_test(3).
 [3,2,1,3]
 [3,2,1,3,2]
 [3,2,1,3,2,1]
 false.
```

Ejemplo: assert

El backtracking no deshace las modificaciones realizadas por assert

En este caso la manera de deshacerlas es con retract(test(_))

¿Para qué puede servir esto?

Predicados NB

- Conjunto de predicados que no son afectados por el backtracking de Prolog
  - NB = non-backtrackable

`nb_setval(+Name,+Value)` ← asocia el átomo Name con el valor Value

`nb_getval(+Name,-Value)` ← obtiene el último valor asociado a Name

Son variables globales al estilo imperativo. Mantienen el valor de la último asignación.

Predicados NB

- Prolog se basa fuertemente en el backtracking
- Programar con predicados que impiden el backtracking está mal visto
  - Es programación imperativa en vez de programación lógica
- Pero programar con variables globales está mal visto incluso en programación imperativa

`nb_setarg(+N,+Term,+Value)` ← setea el N-ésimo argumento del término Term como Value

`arg(+N,+Term,-Value)` ← obtiene el N-ésimo argumento de Term

Si bien sigue siendo imperativo, es un poco más prolijo que utilizar variables globales.

Predicados NB

Utilizando predicados NB, implemente el predicado findall:

```
custom_findall(+Goal,+Template,-Bag)
```

Entrada / Salida

Predicados básicos:

- **`read(-Term)`**
- **`write(+Term)`** (siempre tiene éxito)

Lectura a nivel de caracter:

- **`get_char(-Char)`**
- **`put_char(+Char)`**

Manejo de archivos:

- **`open(+Path,+Mode,-Stream)`** ← Mode puede ser **`read`**, **`write`**, **`append`**, …
- **`close(+Stream)`**
- Luego se utilizan versiones de los predicados **`read`**, **`write`**, **`get`**, **`put`** que toman un stream como primer argumento.

Entrada / Salida

Los predicados de E/S no tienen puntos de backtracking.
Tener espercial cuidado en no perder caracteres leídos.

Ejemplo erróneo:

```
proceso(L) ← L es la lista de caracteres leídos
proceso([]):-
   get_char(C),
   fin(C).
proceso([C|Cs]):-
   get_char(C),
   proceso(Cs).

fin('.').
```

¡El predicado **`proceso`** pierde elementos de la entrada!
Reescribirlo para que funcione correctamente.

Entrada / Salida

- Tenemos versiones de los predicados de E/S con dos argumentos para indicar de dónde leer o en dónde escribir.
- Además es necesario “abrir” los archivos y asociarles un identificador.

```
open(+File, +Mode, -Stream)
read(+Stream, -Term)
get_char(+Stream, -Char)
…
```

Ciclo interactivo

Ejemplo (solamente imprime en la salida la entrada leída)

```
eco:-read(X),eco(X).
eco(salir):-!.
eco(X):-writeln(X),read(Y),eco(Y).
```

Usando repeat:

```
eco:-repeat,read(X),eco(X),!.
eco(salir):-!.
eco(X):-writeln(X),fail.
```

Es posible usar este esqueleto para procesar comandos.