PROGRAMACIÓN DECLARATIVA

PROGRAMACIÓN LÓGICA

Tema PL3: El lenguaje PROLOG, aspectos avanzados

2. Recolección de soluciones

Grado en Ingeniería Informática

URJC

Ana Pradera

Contenido

INTRODUCCIÓN

- Cuando un problema tiene más de una solución, a menudo es muy conveniente poder recolectar todas las posibles soluciones para su posterior tratamiento (visualizarlas, almacenarlas, ordenarlas, transformarlas, contarlas, etc).
- PROLOG ofrece tres predicados predefinidos básicos para recolectar soluciones: `bagof(?T, +Obj, ?L)`, `setof(?T, +Obj, ?L)` y `findall(?T, +Obj, ?L)`.
- Los tres predicados anteriores son ciertos si `L` es una **lista** almacenando **todas** las instancias del término `T` (variable o término compuesto) **que cumplen el objetivo** `Obj`.
- Para ello, construyen el **Árbol de Resolución** correspondiente al objetivo `Obj` retrocediendo automáticamente para generar todas las posibles soluciones.
- Difieren entre ellos en algunos detalles que se explican e ilustran a continuación.

Ejemplo (Programa “Gustos”)

```
% gusta(?X,?Y)
% cierto si a X le gusta el lenguaje Y

gusta(pepa, prolog).
gusta(pepa, haskell).
gusta(pepito, java).
gusta(pepita, prolog).
gusta(pepita, scala).
gusta(pepin, prolog).
gusta(pepin, Algo) :-
        gusta(pepita, Algo).
```

EL PREDICADO bagof(?T, +Obj, ?L)

Cierto si `L` es una lista formada por todas las instancias del término `T` para las que se cumple el objetivo `Obj`.

Ejemplo

```
% Personas a las que les gusta Scala
?- bagof(P, gusta(P, scala), Scaleros).
Scaleros = [pepita, pepin]
```

Si `Obj` no es cierto para ninguna instancia de `T`, el predicado `bagof` falla (la lista `L` de un `bagof` nunca será vacía).

Ejemplo

```
% Personas a las que les gusta Pascal
?- bagof(X, gusta(X, pascal), Pascaleros).
false
```

La lista `L` puede contener soluciones repetidas (*bag* = bolsa) y el orden en el que aparecen las soluciones en la lista *depende del intérprete* (normalmente es el orden en el que PROLOG las encuentra).

Ejemplo

```
 % Personas a las que les gusta Prolog
?- bagof(X, gusta(X, prolog), Prologueros).
Prologueros = [pepa, pepita, pepin, pepin]
```

Ejercicios

Construya el Árbol de Resolución correspondiente a la consulta `?- gusta(X, prolog)` *para comprender por qué pepin aparece dos veces en la lista del ejemplo anterior. Observe también que las soluciones aparecen en* `Prologueros` *de acuerdo con el recorrido en profundidad del Árbol de Resolución.*

`Obj` puede ser un objetivo compuesto por varios subobjetivos colocados entre paréntesis y conectados tanto de forma conjuntiva ("y") como disyuntiva ("o"):

- El predicado predefinido `,/2` se usa en notación infija y representa la *conjunción* de sus dos argumentos, es decir, el objetivo `(Obj1 , Obj2)` será cierto si y sólo si tanto `Obj1` como `Obj2` son ciertos.
- El predicado predefinido `;/2` se usa en notación infija y representa la *disyunción* de sus dos argumentos, es decir, el objetivo `(Obj1 ; Obj2)` será cierto si y sólo si o bien `Obj1` o bien `Obj2` (o ambos) son ciertos.
- Ambos se pueden **extender** a *n* argumentos, es decir, es posible escribir objetivos de la forma `(Obj1 , Obj2 , ... , Objn)` o `(Obj1 ; Obj2 ; ... ; Objn)`, y por supuesto **combinarlos** entre sí, escribiendo, por ejemplo, `((Obj1, Obj2) ; Obj3).`

Ejemplo

```
% Personas a las que les gusta scala Y prolog
?- bagof(X,
         (gusta(X, scala), gusta(X, prolog)),
         L).
L = [pepita, pepin, pepin].


% Personas a las que les gusta scala O prolog
?- bagof(X,
          (gusta(X, scala); gusta(X, prolog)),
          L).
L = [pepita, pepin, pepa, pepita, pepin, pepin]
```

En la última consulta `pepita` aparece dos veces porque le gustan tanto scala como prolog, y pepin aparece tres veces porque le gustan tanto scala como prolog y este último además por dos motivos distintos.

Si `Obj` tiene variables libres (sin valor asignado) distintas a las que aparecen en `T`, el predicado `bagof` dará varias soluciones, una para cada posible valor de la variable libre con algún resultado.

Ejemplo

```
?- bagof(X, gusta(X,Y), L).     % Y está libre
L = [pepa], Y = haskell
L = [pepito], Y = java
L = [pepa, pepita, pepin, pepin], Y = prolog
L = [pepita, pepin], Y = scala
```

Pero cuidado porque no toda variable que aparece en `Obj` (y no en `T`) está libre, puesto que se le puede haber asignado valor antes:

Ejemplo

```
?- Y = scala, bagof(X, gusta(X,Y), L).
 L = [pepita, pepin], Y = scala
```

Es posible cuantificar existencialmente una o más variables de `Obj`, escribiendo `Y1^...Yn^(Obj)`, cuyo significado es "existen `Y1 ... Yn` tales que se cumple `Obj`".

Ejemplo

```
% Personas a las que les gusta algún lenguaje
?- bagof(X, Y^gusta(X,Y), L).
L = [pepa, pepa, pepito, pepita, pepita,
     pepin, pepin, pepin]

% Personas a las que les gustan al menos 2 lenguajes
?- bagof(P,
         L1^L2^(gusta(P,L1),gusta(P,L2),L1 @< L2),
         L).
L = [pepa, pepita, pepin, pepin]
```

La comprobación `L1 @< L2`, L1 menor alfabéticamente que L2, evita la repetición de soluciones en las que L1 y L2 tienen los mismos valores en distinto orden.

El término `T` no tiene por qué ser solo una variable, puede ser un término compuesto.

Ejemplo

```
?- bagof((P,X),
         (gusta(P,X), (X=prolog ; X=scala)),
         D).
D = [(pepa,prolog), (pepita,prolog), (pepita,scala),
     (pepin,prolog), (pepin,prolog), (pepin,scala)]

?- bagof(gustos(Y,L),
         bagof(X,gusta(X,Y),L),
         LL).
LL = [gustos(haskell,[pepa]), gustos(java,[pepito]),
      gustos(prolog,[pepa, pepita, pepin, pepin]),
      gustos(scala,[pepita, pepin])]
```

La última consulta muestra que el objetivo `Obj` puede ser -en general, contener-, a su vez, un recolector de soluciones.

EL PREDICADO `setof(?T, +Obj, ?L)`

Funciona exactamente igual que `bagof` salvo:

1. `L` aparece ordenada de forma ascendente.
2. `L` no contiene repeticiones (*set* = conjunto).

Ejemplo (Uso de setof en comparación con bagof)

```
?- setof(X, gusta(X, prolog), L).
 L = [pepa, pepin, pepita].

?- setof(X, Y^gusta(X,Y), L).
 L = [pepa, pepin, pepita, pepito].

?- setof(X, (gusta(X, scala) ; gusta(X, prolog)), L).
L = [pepa, pepin, pepita].
```

EL PREDICADO `findall(?T, +Obj, ?L)`

Funciona exactamente igual que `bagof` salvo:

1. Todas las variables libres de `Obj` (salvo las que están en `T`) se consideran automáticamente cuantificadas existencialmente.
2. Si `Obj` no es cierto para ninguna instancia de `T`, `findall` devuelve una lista vacía (en lugar de fallar como hace `bagof`).

Ejemplo (Uso de findall en comparación con bagof)

```
?- findall(X, gusta(X,Y), L).
% equivalente a bagof(X, Y^gusta(X,Y), L).
L = [pepa,pepa,pepito,pepita,pepita,
     pepin,pepin,pepin]

?- findall(X, gusta(X, pascal), L).
 % lista vacía cuando no hay ninguna solución
 L = [].
```

Ejercicios (Recolección de soluciones)

Dado el programa ▸ Gustos*, utilice los predicados de recolección de soluciones que considere oportunos para implementar los siguientes predicados:*

1. `numforofos(+Y, ?N)`, *cierto si* `N` *es el número de personas a las que les gusta el lenguaje* `Y`.
2. `lenguajes(-L)`, *cierto si* `L` *contiene el conjunto de todos los lenguajes que gustan a al menos una persona (lista vacía si no hubiese ninguno).*
3. `gustos(-L)`, *cierto si* `L` *es una lista conteniendo todos las listas [P,LL] donde P es una persona y LL es la lista de lenguajes que gustan a P.*

Ejercicios (Recolección de soluciones)

Implemente los siguientes predicados:

1. `concisoYordenado(+L, ?Conj)`, *cierto si* `L` *es una lista no vacía y* `Conj` *es una lista conteniendo los mismos elementos que* `L` *pero sin repeticiones y ordenados de menor a mayor.*
2. `esconjunto(+L)`, *cierto si L es un conjunto, es decir, L es una lista que no contiene elementos repetidos.*
3. `interseccion(+L1, +L2, L)`, *cierto si L es un conjunto conteniendo todos los elementos comunes a L1 y L2. Tenga en cuenta que L no debe contener elementos repetidos, y que si L1 y L2 no tienen elementos comunes, L debe ser igual a la lista vacía.*
4. `union(+L1, +L2, L)`, *cierto si L es un conjunto conteniendo todos los elementos que están en L1 o bien en L2 (o en ambos). Tenga en cuenta que L no debe contener elementos repetidos, y que si L1 y L2 son vacías, L debe ser también vacía.*

Ejercicios (Recolección de soluciones)

- *Ejercicio nº 1, apartado 1.1, de la Práctica de* PROLOG *nº 4.*
- *Ejercicio nº 2, apartados 2.1, 2.2 y 2.3, de la Práctica de* PROLOG *nº 4.*

BIBLIOGRAFÍA

- L. Sterling and E. Shapiro. *The Art of Prolog*. The MIT Press, Cambridge, Mass., second edition, 1994.
- W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, fifth edition, 2003.
- I. Bratko.*Prolog Programming for Artificial Intelligence*. Addison-Wesley, Reading, Massachusetts, third edition, 2001.
- J. Lloyd. *Foundations of Logic Programming*, (Second Edition). Springer-Verlag, 1987.
- R. O'Keefe. *The Craft of Prolog*. The MIT Press, Cambridge, MA, 1990.
- U. Nilsson and J. Maluszynski. Logic, Programming and Prolog. John Wiley & Sons Ltd, 1996.
- SWI-Prolog, entorno de programación en Prolog de dominio público.
- comp.lang.prolog. Faq

© 2022 Ana Pradera Gómez
Algunos derechos reservados
Este documento se distribuye bajo la licencia
“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,
disponible en
https://creativecommons.org/licenses/by-sa/4.0/deed.es