Représentation de la Connaissance

Répétition 7

7 novembre 2006

Astuce du jour

Le prédicat prédéfini =.. permet de construire et de déconstruire des termes.

```
Term =.. L
```

est vrai si L est une liste contenant le foncteur principal de `Term` suivi de ses arguments. Les exemples suivants illustrent son fonctionnement :

```
?- f(a,b) =.. L.
L=[f,a,b]
?- T =.. [rectangle,3,5].
T=rectangle(3,5)
```

Pourquoi aurait-on envie de décomposer un terme en ses composants, i.e. son foncteur et ses arguments ? Pourquoi construire un nouveau terme à partir d'un foncteur et d'arguments donnés ? Illustrons la nécessité de ce procédé à partir d'un exemple simple.

Exemple d'utilisation

Considérons un programme manipulant des figures géométriques. Celles-ci peuvent être représentées par des termes dont le foncteur indique le nom de la figure et les arguments les différents paramètres de taille de la figure. Soit par exemple :

```
square(Side)
triangle(Side1,Side2,Side3)
circle(R)
```

Supposons qu'on veuille écrire une fonction d'élargissement des figures :

```
enlarge(Fig, Facteur, FigNew)
```

où `Fig` et `FigNew` sont des figures géométrique du même type et les paramètres de `FigNew` sont ceux de `Fig` multipliés par `Facteur`. Pour la simplicité, on suppose que les paramètres de `Fig` sont instanciés, ainsi que `Facteur`. Une manière de programmer le prédicat serait la suivante :

```
enlarge(square(A),F,square(A1)) :- A1 is A*F.
enlarge(circle(R),F,circle(R1)) :- R1 is R*F.
enlarge(rectangle(A,B),F,rectangle(A1,B1)) :- A1 is A*F, B1 is B*F.
...
```

Il faudra donc écrire une ligne par figure géométrique ! Pour les figures à un paramètre, on pourrait tenter de solutionner le problème à l'aide du code suivant :

```
enlarge(Type(Par),F,Type(Par1)) :- Par1 is F*Par.
```

Malheureusement ce code n'est pas correct car Prolog n'accepte normalement que des atomes comme foncteur. La méthode correcte, et fonctionnant pour un nombre quelconque d'arguments, consiste naturellement à utiliser le prédicat prédéfini =..

```
enlarge(Fig,F,Fig1) :-
   Fig =.. [Type|Parameters],
   multiplylist(Parameters,F,Parameters1),
   Fig1 =.. [Type|Parameters2].

multiplylist([],_,[]).
multiplylist([X|L],F,[X1|L1]) :-
   X1 is F*X, multiplylist(L,F,L1).
```

Exercices sur le prédicat prédéfini =..

1. `apply_pred`
Définir le prédicat `apply_pred(pred,arg)` vrai si et seulement si `pred(arg)` est vrai. Ainsi `apply(number,1)` est vrai mais `apply(number,a)` est faux.

2. `filter`

Définir le prédicat `filter(l1,pred,l2)` vrai si et seulement si `l2` est la liste des éléments de `l1` qui satisfont le prédicat `pred`.

Exemple(s) d'utilisation:

```
filter([1,2,a,b,3,d,f,4],number,Ys).
Ys=[1,2,3,4];
No.
```

3. `and_map`

Définir le prédicat `and_map(l,pred)` vrai si et seulement si tous les éléments de la liste `l` vérifient le prédicat `pred`.

Exemple(s) d'utilisation:

```
and_map([1,2,3,4,5,6],number).
Yes
and_map([1,2,3,a,5,6],number).
No
```

Exercices sur les graphes

Correction de l'exercice proposé

4. Labyrinthe

Considérons le labyrinthe suivant

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Ecrire un programme Prolog permettant de trouver un chemin de 1 vers 11.