Représentation de la Connaissance

Répétition 1

26 septembre 2006

1 Avant propos

Informations pratiques

- Assistant : François Van Lishout
- `vanlishout@montefiore.ulg.ac.be`, 04/3662619, bureau I82a.
- http ://www.montefiore.ulg.ac.be/∼vanlishout
- Toute initiative de l'étudiant est la bienvenue dans ce cours.
- N'hésitez pas à venir me trouver pour parler de vos programmes (bons ou mauvais).

Astuce du jour

Pour charger un fichier intitulé "nom_fichier.pl", exécuter la commande

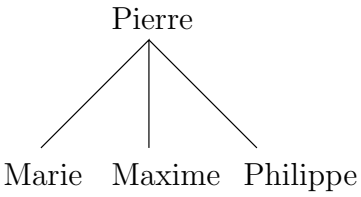
[nom_fichier].

2 La programmation logique avec PROLOG

Les bases de PROLOG

Exercices introductifs

1. Soit l'arbre généalogique (simple)



décrit par le programme ci-dessous :

```
homme(pierre).
homme(maxime).
homme(philippe).
femme(marie).
pere(pierre,maxime).
pere(pierre,philippe).
pere(pierre,marie).
```

Définir alors les prédicats suivants
- `enfant(X,Y)` vrai si `X` est un enfant de `Y` ;
- `fils(X,Y)` vrai si `X` est un fils de `Y` ;
- `fille(X,Y)` vrai si `X` est une fille de `Y` ;
- `frere_ou_soeur(X,Y)` vrai si `X` est frère ou soeur de `Y` ;
- `frere(X,Y)` vrai si `X` est frère de `Y` ;
- `soeur(X,Y)` vrai si `X` est soeur de `Y`.

2. `last2`

Définir un prédicat `last2(A,Ls)` vrai lorsque `A` est le dernier élément de la liste non vide `Ls`, et faux sinon. On demande que, si `A` n'est pas complètement instancié mais que `Ls` l'est, l'instance de `A` telle que `last2(A,Ls)` est vrai soit générée.

3. `member2`

Définir un prédicat `member2(A,Ls)` vrai lorsque l'objet `A` est un élément de la liste `Ls`, et faux sinon. On demande que, dans le cas où `A` n'est pas complètement instancié mais que `Ls` l'est, les différentes instances de `A` telles que `member2(A,Ls)` est vrai soient générées.

4. `fact`

Définir un prédicat `fact(A,B)` vrai si et seulement si `B` est la factorielle du naturel (non strictement) positif `A`. Ce prédicat pourra aussi servir pour calculer la factorielle d'un nombre.

5. `all-equal`

Définir un prédicat `all-equal(Ls)` vrai si et seulement si `Ls` est une liste dont tous les éléments sont égaux.

6. `length2`

Définir un prédicat `length2(Ls,N)`, où `Ls` est une liste et `N` est un naturel, vrai si et seulement si la liste `Ls` contient exactement `N` éléments. On demande que, dans le cas où `N` n'est pas instancié mais que `Ls` l'est, l'instance de `N` telle que `length2(Ls,N)` est vrai soit générée.

PROLOG et les listes

Exercices simples

7. `insere-elem`

Définir un prédicat `insere-elem(Xs,E,Ys)`, vrai si et seulement si la liste `Ys` est la liste `Xs` dans laquelle l'élément `E` a été inséré après chaque élément de `Xs`. Dans le cas où `Ys` n'est pas complètement instancié, on demande que l'instance qui satisfait le prédicat soit retournée.

Exemple(s) d'utilisation:

```
insere-elem([titanic,a,coule],stop,X).
X = [titanic,stop,a,stop,coule,stop];
No
```

8. `regroupe`

Définir un prédicat `regroupe(Xs,Ys)`, vrai si et seulement si la liste `Ys` est la liste des éléments de `Xs`, groupés par listes de deux éléments. On demande que dans le cas où `Ys` n'est pas complètement instancié, l'instance vérifiant le prédicat soit retournée.

Exemple(s) d'utilisation:

```
regroupe([cochon,truie,taureau,vache,cheval,jument],X).
X = [[cochon,truie],[taureau,vache],[cheval,jument]];
No
```

Exercices proposés

9. `remove_last`

Définir un prédicat `remove_last(Xs,Ys)`, où `Xs` est une liste non vide et `Ys` une liste, vrai si et seulement si `Ys` est `Xs` dont on a retiré le dernier élément. On demande que, dans le cas où `Ys` n'est pas complètement instancié mais que `Xs` l'est, l'instance de `Ys` telle que `remove_last(Xs,Ys)` est vrai soit générée.

10. `rep`

Définir un prédicat `rep(Xs,Ys)`, vrai si `Ys` est la liste `Xs` dont on a retiré les répétitions, en supposant que la liste `Xs` est triée. On demande que dans le cas où `Ys` n'est pas complètement instancié mais que `Xs` l'est, l'instance de `Ys` telle que `rep(Xs,Ys)` est vrai soit générée.

Exemple(s) d'utilisation:

```
rep([1,1,1,2,2,2,2,4,4,6,7],X).
X = [1,2,4,6,7];
No
```

11. subset_bis

Définir un prédicat `subset_bis(Ss,Ls)` vrai lorsque `Ss` est une liste d'éléments de `Ls`, dans le même ordre d'apparition. On demande que dans le cas où `Ss` n'est pas complètement instancié mais que `Ls` l'est, les différentes instances de `Ss` telles que `subset_bis(Ss,Ls)` est vrai soient générées.

Exemple(s) d'utilisation:

```
?- subset_bis([1,3,4],[0,1,2,3,4,5]).
Yes
```

12. sublist

Définir un prédicat `sublist(Ss,Ls)` vrai lorsque la liste `Ss` est une sous-liste de la liste `Ls`, et faux sinon. On demande que, dans le cas où `Ss` n'est pas complètement instancié mais que `Ls` l'est, les différentes instances de `Ss` telles que `sublist(Ss,Ls)` est vrai soient générées.

Exemple(s) d'utilisation:

```
sublist([2,3],[1,2,3,4]).
Yes
sublist([2,4],[1,2,3,4]).
No
```

13. inc_sublist

Définir un prédicat `inc_sublist(Ls,Is)` vrai si et seulement si `Is` est la plus longue sous-liste croissante de la liste de nombres `Ls`. Dans le cas où `Is` n'est pas complètement instancié mais que `Ls` l'est, on demande d'écrire deux versions différentes :

- La première renverra une seule instance de `Is` telle que `inc_sublist(Ls,Is)` est vrai.
- La deuxième renverra une et une seule fois chacunes des instances de `Is` telles que `inc_sublist(Ls,Is)` est vrai.

Exemple(s) d'utilisation:

```
inc_sublist([1,2,1,4,5,6,4,3,7],X).
X = [1,4,5,6]; No
inc_sublist([1,2,3,1,3,6,1,4,8],X).
% 1ere version :     % 2eme version :
X = [1,2,3]; No      X = [1,2,3]; X = [1,3,6]; X = [1,4,8]; No
```