Université Paris 7 - Master 1 Informatique - Prolog et Programmation avec contraintes

Examen du 11 septembre 2006 - Durée : 2 heures

Informations : Tous les documents reliés sont autorisés. Le barème est donné à titre indicatif et peut être modifié.

Exercice 1 (**4 points**)

Le tableau suivant contient 4 paires de programmes Prolog: (p1, q1),...,(p4,q4).

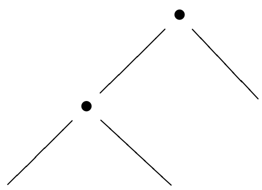
Pour chaque paire (pi,qi) exhiber (s'il existe) un but gi tel que pi et qi fournissent des résultats différents pour gi (ceci peut signifier soit que l'un réussit et l'autre echoue, soit que les ensembles des résultats (unifications) obtenus pour une variable du but sont différents). Motiver vos réponses.

p1: `p(a):-!.` `p(b).`	q1: `p(a).` `p(b):-!.`
p2: `p(a).` `p(b).`	q2: `p(a).` `p(b):-!.`
p3: `p(a).` `q(b).` `r(X):-not(p(X)),q(X).`	q3: `p(a).` `q(b).` `r(X):-q(X),not(p(X)).`
p4: `p(a).` `q(a).` `q(b).` `r(X):-p(X),!,q(X).`	q4: `p(a).` `q(a).` `q(b).` `r(X):-p(X),q(X),!.`

Exercice 2 (**6 points**)

On définit une famille de termes pour représenter les arbres binaires sans étiquettes en Prolog: un arbre est soit la constante `f` (une feuille), soit un terme de la forme `n(t1,t2)` (un noeud dont les fils sont`t1` et `t2`), où `t1` et `t2` sont deux arbres.

Par exemple, le terme `n(n(f,f),f)` est un arbre, que graphiquement on peut représenter comme suit:



La *hauteur* d'une feuille est 0, et la hauteur d'un noeud est le max des hauteurs de ses fils plus 1. La *taille* d'une feuille est 0, et la taille d'un noeud est la somme des tailles de ses fils plus 1.

- Définir deux prédicats `hauteur(+arbre,-res)`, `taille(+arbre,-res)` pour calculer hauteur et taille d'un arbre donné.
- Un *parcours* est une liste de `g` et `d` (pour "gauche" et "droite"). Définir un prédicat `p(+arbre,-parcours)` qui engendre, pour un arbre donné, tous les parcours qui mènent de la racine aux feuilles (qu'on appelle *parcours complets*). Par exemple:

```
  ?-  parcours(n(n(f,f),f),X).
X = [g,g] ? ;
X = [g,d] ? ;
X = [d] ? ;
no
```

- Un arbre binaire est *équilibré* si la difference entre les longueurs de deux parcours complets quelquonques dans l'arbre (c.à.d. la difference entre les longueurs de deux branches quelquonques) est inferieure ou égale à 1. Définir un prédicat `equilibre(+arbre)` qui réussit si et seulement si `arbre` est équilibré.

- Définir un prédicat `greffe(+arbre1, +arbre2, +parcours, -arbre3)` qui réussit si `parcours` est un parcours complet pour `arbre1`, et dans ce cas donne comme résultat (en `arbre3`), `arbre1` dans lequel `arbre2` est greffé à la feuille qui se trouve au bout de `parcours`. Par exemple:

  ```
  ?-  greffe(  n(n(f,f),f)  , n(f,f) , [d] , R).
  R = n(n(f,f),n(f,f)) ? ;
  no
  ```

  Si `parcours` n'est pas complet pour `arbre1`, le prédicat `greffe` echoue.