M1 Informatique – Paris Diderot - Paris 7
Programmation Logique et par Contraintes

Examen partiel du 20 octobre 2016 - Durée: 1h50

Documents autorisés; le barème est donné à titre indicatif.

Exercice 1 (**5 points**) Pour chacune des requêtes suivantes, donner le résultat renvoyé par l'interpréteur Prolog (sans justifier) :

1. `f(X,a,Z)=f(a,Z,b).`
2. `f(X,a,Z)=f(a,Z,B).`
3. `f(a,b)==f(A,b).`
4. `fail -> fail;!.`
5. `fail; (!,fail).`
6. `[X|[Y|[Z|L]]]=[a,b,[c,d]].`

Dans les questions suivantes, on suppose que le fichier `ex.pl` ci-dessous ait été compilé:

```
/********* ex.pl *********/
a(0,1).
b(X,1):-X=0.
c(X,1):-X=:=0.
d(X,1):-X==0.
/***********************/
```

7. `a(1-1,Y).`
8. `b(0,Y).`
9. `c(1-1,Y).`
10. `d(1-1,Y).`

Exercice 2 (**4 points**)

1. Écrire un prédicat `maximum(+L,-M)` qui calcule le plus grand élément d'une liste non vide d'entiers:

   ```
   [eclipse 1]:  maximum([1,3,2],M).
   M = 3
   Yes (0.00s cpu)
   ```

2. Écrire un prédicat `max_liste(+L,-M)` qui prend une liste non vide dont les éléments sont des listes et renvoie la liste de longueur maximale:

   ```
   [eclipse 2]:  max_liste([[a,b],[1,2,3,4],["aa",a]],L).
   L = [1, 2, 3, 4]
   Yes (0.00s cpu)
   ```

   Vous pouvez utiliser sans le définir le prédicat `length(+L,-N)` qui renvoie la longueur d'une liste.

3. Nous allons à présent généraliser l'opération de sélection des prédicats `maximum` et `max_liste`: en utilisant le prédicat infixe `=..` écrire un prédicat `selection(+Liste, +Max, -Result)` qui prend une liste non vide et le nom d'un prédicat de comparaison à trois argument, applicable aux éléments de la liste et renvoie le plus grand élément de `Liste` relativement au prédicat `p` passé en `Max`, en supposant que `p(+X,+Y,-Z)` renvoie en `Z` le maximum entre `X` et `Y`.

   Par exemple, après avoir compilé les définitions suivantes:

   ```
   maximum(X,Y,Z):- X>=Y -> Z=X;Z=Y.
   max_long(X,Y,Z):- length(X,N),length(Y,M), N>=M -> Z=X;Z=Y.
   ```

on doit avoir :

```
[eclipse 3]: selection([1,3,2],maximum,M).
M = 3
Yes (0.00s cpu)

[eclipse 4]: selection([[a,b],[1,2,3,4],["aa",a]],max_long,L).
L = [1, 2, 3, 4]
Yes (0.00s cpu)
```

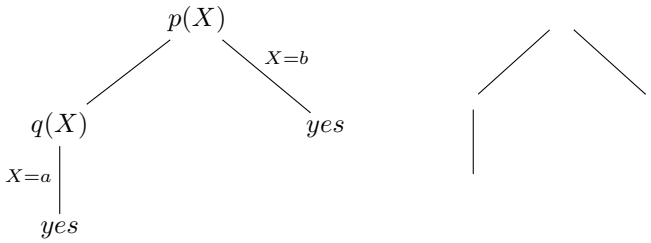
Exemple d'utilisation de `=..` : `Q=..[p,X,Y,Z]` unifie la variable `Q` et le terme `p(X,Y,Z)`.

Exercice 3 (**5 points**)

Si t est un arbre de dérivation, on appelle *squelette de t* l'arbre ayant la même strucure, mais sans aucune étiquette. Par exemple, l'arbre de dérivation du but `p(X)` pour le le programme

```
q(a).
p(X):-q(X).
p(b).
```

et son squelette sont dessinés ci-dessous:



1. On considère le programme `p(a).p(b).q(a).q(b).r(a).r(b).` Dessinez les squelettes des arbres de dérivations des buts (sans motiver):
   (a) `p(X),q(Y),r(Z).`
   (b) `p(X),q(Y),r(Z),!`
   (c) `p(X),q(Y),!,r(Z).`
   (d) `p(X),!,q(Y),r(Z).`
   (e) `p(X),r(X),q(X).`
2. Soit n un entier. Donner un programme prolog et un but tels que le squelette de l'arbre de dérivation correspondant soit:
   (a) l'arbre binaire complet de hauteur n.
   (b) une branche (un fil) de hauteur n.

Exercice 4 (**6 points**) *Pour chaque question de cet exercice, vous disposez des prédicats décrits dans les questions précédentes, même si vous ne les avez pas implementés.*

Si `l` est une liste d'entiers, une *sous-liste* de `l` est une séquence d'éléménts contigus de `L`. Une sous-liste de `L` est *triée* si ses éléments sont disposés dans l'ordre croissant, de gauche à droite. Une sous-liste triée de `l` est *maximale* elle ne ne peut être étendue à gauche ni à droite, tout en restant triée (l'exemple ci-dessous illustre cette définition). Considérons le jeu combinatoire dont les positions sont des listes d'entiers, et une liste `l` est atteignable à partir d'une liste `h` si `l` est obtenue par effacement d'une sous-liste triée maximale de `h`.

Par exemple, à partir de la liste `[1,2,1,2,3,4,3,4]`, trois listes sont atteignables:

- `[1,2,3,4,3,4]`, obtenue par effacement de la sous-liste triée maximale `[1,2]`.
- `[1,2,3,4]`, obtenue par effacement de la sous-liste triée maximale `[1,2,3,4]`.
- `[1,2,1,2,3,4]`, obtenue par effacement de la sous-liste triée maximale `[3,4]`.

Le joueur qui vide la liste gagne.

1. Écrire un prédicat `sous_listes(+L,-H)` qui prend une liste d'entiers `L` et renvoie la liste des sous-listes triées maximales de L.

   Exemple:

   ```
   eclipse 1]: sous_listes([1,2,1,2,3,4,3,4],H).
   H = [[1, 2], [1, 2, 3, 4], [3, 4]]
   Yes (0.00s cpu)
   ```

2. Écrire un prédicat `aplatir(+L,-H)` qui prend une liste de listes et renvoie la liste obtenue en concatenant les listes contenues dans `L`. Vous pouvez utiliser sans le définir le prédicat `concat(+L,+,G,-H)` qui renvoie en `H` la concaténation des listes `L` et `G`.

   ```
   [eclipse 2]: aplatir([[1, 2], [1, 2, 3, 4], [3, 4]], L).
   L = [1, 2, 1, 2, 3, 4, 3, 4]
   Yes (0.00s cpu)
   ```

3. Écrire un prédicat `enlever(+L,-H)` qui enléve un élément de `L`:

   ```
   [eclipse 3]: enlever([[1, 2], [1, 2, 3, 4], [3, 4]], L).
   L = [[1, 2, 3, 4], [3, 4]]
   Yes (0.00s cpu, solution 1, maybe more) ? ;
   L = [[1, 2], [3, 4]]
   Yes (0.00s cpu, solution 2, maybe more) ? ;
   L = [[1, 2], [1, 2, 3, 4]]
   Yes (0.00s cpu, solution 3, maybe more) ? ;
   No (0.00s cpu)
   ```

4. Écrire un prédicat `move(+L,-H)` qui renvoie toutes les listes atteignables à partir de `L` dans le jeu:

   ```
   [eclipse 57]: move([1,2,1,2,3,4,3,4],L).
   L = [1, 2, 3, 4, 3, 4]
   Yes (0.00s cpu, solution 1, maybe more) ? ;
   L = [1, 2, 3, 4]
   Yes (0.00s cpu, solution 2, maybe more) ? ;
   L = [1, 2, 1, 2, 3, 4]
   Yes (0.00s cpu, solution 3, maybe more) ? ;
   No (0.00s cpu)
   ```

5. Écrire un prédicat `gagne(+L)` qui réussit si `L` est une position gagnante, en utilisant la stratégie force brute.

6. La position `[1,2,1,2,3,4,3,4]` est-elle gagnante?